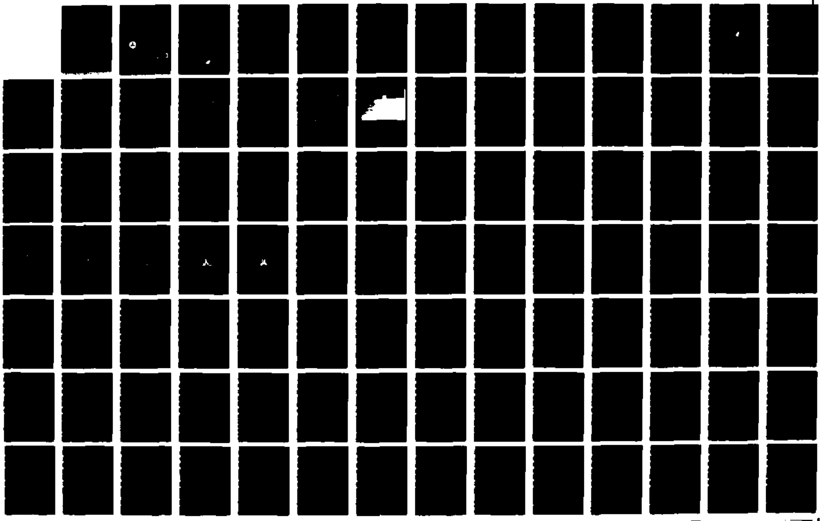
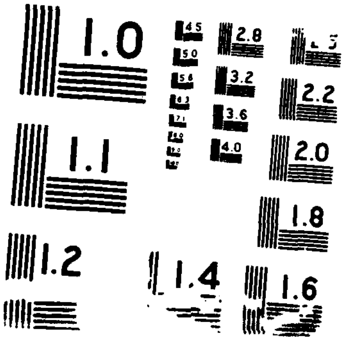
AD-A194 665 PROBABILITIES OF DETECTION AND RECOGNITION OF FLASHING 1/2
LIGHTS ON ROLLING BUOYS(U) COAST GUARD RESEARCH AND
DEVELOPMENT CENTER GROTON CT D M BROWN AUG 87
UNCLASSIFIED CGR/DC-06/87 USCG-D-10-88 F/G 17/7.2 NL




1.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.2
1.4
1.6

2

Report No. CG-D-10-88

PROBABILITIES OF DETECTION AND RECOGNITION OF FLASHING LIGHTS ON ROLLING BUOYS

DANIEL M. BROWN

U.S. COAST GUARD RESEARCH AND DEVELOPMENT CENTER
AVERY POINT, GROTON, CONNECTICUT 06340-6096

INTERIM REPORT
AUGUST 1987

This document is available to the U.S. public through the
National Technical Information Service, Springfield, Virginia 22161

Prepared for:
U.S. Department Of Transportation
United States Coast Guard
Office of Engineering and Development
Washington, DC 20593

NOTICE

This document is disseminated under the sponsorship of the Department of Transportation in the interest of information exchange. The United States Government assumes no liability for its contents or use thereof.

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the object of this report.

The contents of this report reflect the views of the Coast Guard Research and Development Center, which is responsible for the facts and accuracy of data presented. This report does not constitute a standard, specification, or regulation.

SAMUEL F. POWEL, III
Technical Director
U.S. Coast Guard Research and Development Center
Avery Point, Groton, Connecticut 06340-6096

Technical Report Documentation Page

1. Report No.
CG-D-10-88

2. Government Accession No.
ADA194665

3. Recipient's Catalog No.

4. Title and Subtitle

PROBABILITIES OF DETECTION AND RECOGNITION
OF FLASHING LIGHTS ON ROLLING BUOYS

5. Report Date
AUGUST 1987

6. Performing Organization Code

7. Author(s)
Daniel M. Brown

8. Performing Organization Report No.
CGR&DC 06/87

9. Performing Organization Name and Address
U.S. Coast Guard
Research and Development Center
Avery Point
Groton, Connecticut 06340-6096

10. Work Unit No. (TRAIS)

11. Contract or Grant No.

12. Sponsoring Agency Name and Address
Department of Transportation
U.S. Coast Guard
Office of Engineering and Development
Washington, D.C. 20593

13. Type of Report and Period Covered
INTERIM

14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

This report shows how buoy motion affects a mariner's ability to detect and recognize standard Coast Guard buoy lights. Buoy roll reduces the effective intensity of the light in the mariner's direction, which results in a detection distance much shorter than the currently used nominal range. Data for this analysis were provided by recording motion of two buoys with a specially designed optical device. The data were analyzed to predict variation of light intensity in a mariner's direction over time and, furthermore, to predict probabilities of detecting and recognizing the flash characteristics of the light. Results show increased detection range with increased vertical divergence, reduced detection range when buoy roll period and flash period are similar, and detection distances as short as 30% of published nominal range. (Keywords.)

17. Key Words
Buoy Motion
Probability of Detection
Probability of Recognition
Response Amplitude Operator
Buoy Lights
Aids to Navigation

18. Distribution Statement
Document is available to the U.S. public through the National Technical Information Service, Springfield, Virginia 22161

19. Security Classif. (of this report)
UNCLASSIFIED

20. SECURITY CLASSIF. (of this page)
UNCLASSIFIED

21. No. of Pages

22. Price

METRIC CONVERSION FACTORS

Approximate Conversions to Metric Measures

Symbol	When You Know	Multiply By	To Find	Symbol
		LENGTH		
in	inches	*2.5	centimeters	cm
ft	feet	30	centimeters	cm
yd	yards	0.9	meters	m
mi	miles	1.6	kilometers	km
		AREA		
in^2	square inches	6.5	square centimeters	cm^2
ft^2	square feet	0.09	square meters	m^2
yd^2	square yards	0.8	square meters	m^2
mi^2	square miles	2.6	square kilometers	km^2
	acres	0.4	hectares	ha
		MASS (WEIGHT)		
oz	ounces	28	grams	g
lb	pounds	0.45	kilograms	kg
	short tons (2000 lb)	0.9	tonnes	t
		VOLUME		
tsp	teaspoons	5	milliliters	ml
tbsp	tablespoons	15	milliliters	ml
fl oz	fluid ounces	30	milliliters	ml
c	cups	0.24	liters	l
pt	pints	0.47	liters	l
qt	quarts	0.95	liters	l
gal	gallons	3.8	liters	l
ft^3	cubic feet	0.03	cubic meters	m^3
yd^3	cubic yards	0.76	cubic meters	m^3
		TEMPERATURE (EXACT)		
°F	Fahrenheit temperature	5/9 (after subtracting 32)	Celsius temperature	°C

*1 in = 2.54 (exactly). For other exact conversions and more detailed tables, see NBS Misc. Publ. 286, Units of Weights and Measures. Price $2.25. SD Catalog No. C13.10.286.

Approximate Conversions from Metric Measures

Symbol	When You Know	Multiply By	To Find	Symbol
		LENGTH		
mm	millimeters	0.04	inches	in
cm	centimeters	0.4	inches	in
m	meters	3.3	feet	ft
m	meters	1.1	yards	yd
km	kilometers	0.6	miles	mi
		AREA		
cm^2	square centimeters	0.16	square inches	in^2
m^2	square meters	1.2	square yards	yd^2
km^2	square kilometers	0.4	square miles	mi^2
ha	hectares (10,000 m^2)	2.5	acres	
		MASS (WEIGHT)		
g	grams	0.035	ounces	oz
kg	kilograms	2.2	pounds	lb
t	tonnes (1000 kg)	1.1	short tons	
		VOLUME		
ml	milliliters	0.03	fluid ounces	fl oz
l	liters	0.125	cups	c
l	liters	2.1	pints	pt
l	liters	1.06	quarts	qt
l	liters	0.26	gallons	gal
m^3	cubic meters	35	cubic feet	ft^3
m^3	cubic meters	1.3	cubic yards	yd^3
		TEMPERATURE (EXACT)		
°C	Celsius temperature	9/5 (then add 32)	Fahrenheit temperature	°F

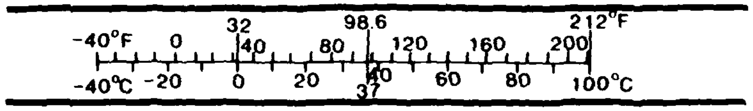


TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

[BLANK]

1. INTRODUCTION

The United States Coast Guard maintains approximately 50,000 Federal aids to navigation on United States waterways. The Coast Guard publishes information on these navigational aids in Light Lists and Notices to Mariners. For approximately 4100 lighted buoys, this information includes the flash characteristic of the light (Morse A, quick flash, etc.) and nominal range.

A typical omnidirectional aid to navigation lantern consists of fresnel drum lens with a vertical filament lamp at its center. The drum lens refracts light toward the optical plane (plane perpendicular to lantern's vertical axis), thus increasing optical plane intensity over that provided by a bare lamp alone. Intensity peaks in the optical plane, and rapidly falls off to zero above and below the optical plane with an approximately Gaussian profile, as shown in Figure 1. Profile full width at half maximum, called vertical divergence, is about 5 degrees for the Coast Guard 155mm buoy lantern.

Currently, nominal range is the standard performance measure of all navigational lights and indicates the distance at which most observers will detect a light. Nominal range calculations are based on optical plane peak intensity. These calculations are appropriate for fixed structures where the optical plane is constantly aligned with the mariner's line of sight. Each flash of the light produces the same peak intensity in the mariner's direction and results in a constant detectability.

However, nominal range calculations are inappropriate for floating aids to navigation where the optical plane pitches and rolls with the buoy. As the buoy rolls, intensity in the mariner's direction continually changes between its peak value and zero, as illustrated in Figure 2. This variation in intensity over time causes the detection distance of the buoy light to change over time. The detection distances may be represented as a probability distribution function with the most probable detection distance being significantly less than the calculated nominal range value.

After detecting a lighted buoy, the mariner must identify its characteristic. This requires detecting enough flash pulses to distinguish it from other possible lighted buoys. Buoy motion may cause some flashes to go undetected, making the recognition task more difficult.

Solutions to the problem of detecting and recognizing flashed buoy lights are not apparent, particularly when no method exists to quantify the problem or evaluate proposed solutions. Greater vertical divergence produces increased intensities at angles off the optical plane, but decreased intensity on and near the optical plane. Greater flash characteristic duty cycle produces increased probability of seeing flashes, but decreased battery life. Redesigned buoys with natural roll frequencies

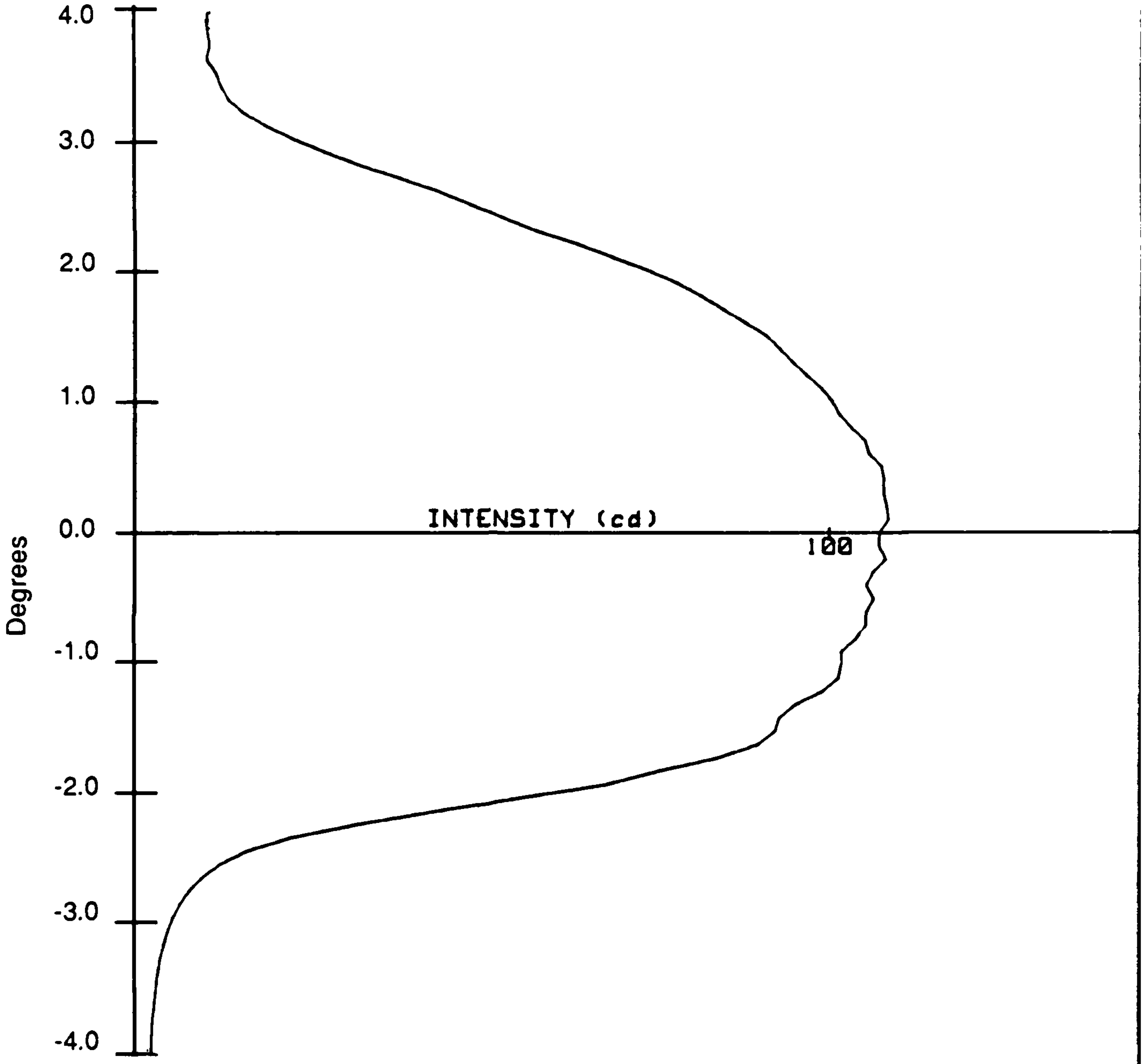


FIGURE 1. Typical Vertical Intensity Profile of 155mm Lantern

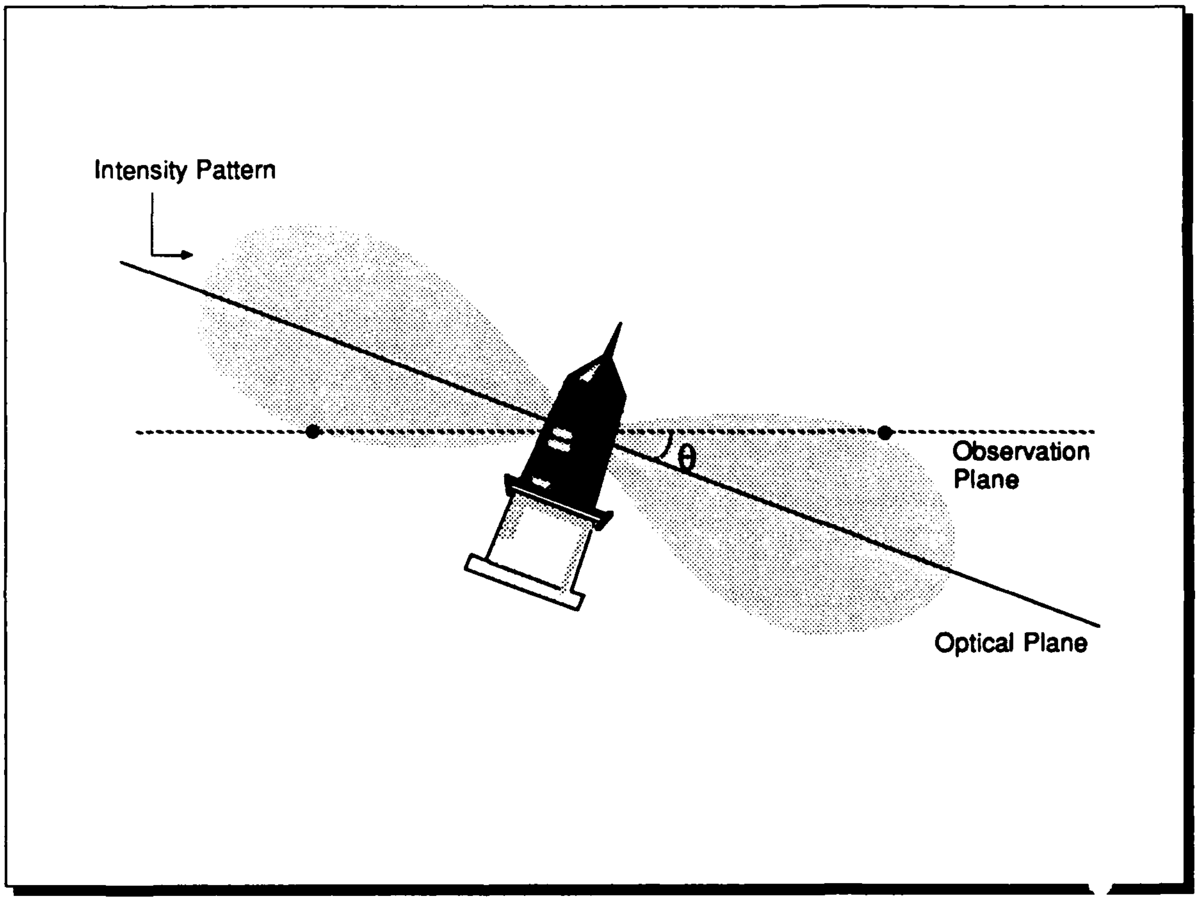


FIGURE 2. Effect of Buoy Angle on Observed Intensity

significantly different from predominant wave frequencies may achieve reduced buoy motion and increased detection range of the light. A method to measure the degree to which these may solve the problem has not been available until now.

This study assesses the impact of buoy motion on a mariner's ability to detect and recognize standard Coast Guard buoy lights. It proposes a new measure of buoy light performance and introduces a new method to evaluate different buoy designs, lantern designs, and flash characteristics. The study shows buoy light detectability depends on vertical divergence, flash characteristic, buoy roll amplitudes, and other variables.

2. HISTORY

Feingold, Ritter, and Tozzi (1977) proposed a theoretical model for predicting probabilities of detecting a buoy light given the probability density function of buoy roll angle. They defined probability of detection as the integral (over all angles) of the product of two probability density functions: a) the probability of detection distribution versus angle and b) the buoy roll angle probability distribution. The probability of detection distribution versus angle was derived from simple geometric considerations.

Hirsch (1982) used the formulation of Feingold et. al. (1977) to assess the influence of vertical divergence on POD. He reported a significant increase in probability of detection with increasing vertical divergence, but did not take into account the necessary decrease in optical plane intensity due to spreading the flux over wider divergence angles.

There have been other attempts by Coast Guard engineers to model detection probability of buoy lights, but none have been widely accepted. Most efforts and debates have centered on vertical divergence of buoy lanterns. Flash characteristic is generally not considered in probability of detection models, although it is the simplest and least expensive design variable to change in the lighted buoy system.

3. A NEW APPROACH

Previous efforts to predict probabilities of detection of rolling buoys did not consider observation time, distance from buoy or flash characteristic. Intuitively, one expects probability of detection to increase the longer a mariner looks in the direction of the buoy. One also expects distance from the buoy and flash characteristic duty cycle to affect detection probability.

This author's approach was to determine the number of detected flashes of a light on an actual rolling buoy for various distances and observation times. Detection and recognition probabilities were calculated from a time series history of intensity in the horizontal plane for a flashed light on a rolling buoy. Probability of detection (POD) and probability of recognition (POR) were modeled as functions of flash characteristic, lantern divergence, observation time, observation distance, and buoy roll.

3.1 Definitions

The following definitions apply to this report.

Effective Luminous Intensity. - Luminous intensity is the luminous flux per unit solid angle, measured in units of candela. The effective luminous intensity (or equivalent fixed intensity) of a flashing light is the intensity of a fixed-on light that produces the same luminosity as the flashing light. The apparent reduction of intensity for flashing lights is due to the eye's time response. In this report effective intensity, I_e, is calculated from a flash's instantaneous intensity, I(t), and peak intensity, I_p, using the Schmidt-Clausen method:

$$I_p \,/\, I_e = 1 + 0.2 \;(I_p \,/ \int I(t)\; dt \;) \qquad (1)$$

where the integral is over the flash duration.

Nominal Range. - Maximum distance, D, a light can be detected based on a light's effective luminous intensity, a meteorological visibility of 10 nautical miles (19 kilometers), and a threshold of illuminance at the eye of 0.67 sea-mile-candela (0.2 microlux). It is calculated from Allard's Law:

$$0.67 = I_e\, T^D \,/\, D^2 \qquad (2)$$

where the atmospheric transmissivity, T, is determined from a meteorological visibility, V, of 10 nautical miles by:

$$T = 0.05^{1/V} = 0.74 \qquad (3)$$

Flash Period. - The length of time to complete one cycle of a flash characteristic. Example: an FL 6 (.6) characteristic has a flash period of 6 seconds.

Flash Pulse. - Refers to a single period of time when the lamp is lit. Example: an I QK FL characteristic has six flash pulses per flash period. A steady fixed on light that is fading on and off due to buoy motion has only one flash pulse.

Observation Event. - A single event wherein a mariner at a specified distance looks in the direction of the buoy for a finite but continuous time period.

Observation Time. - The period of time in seconds a mariner looks in direction of buoy.

Probability of Detection (POD). - The proportion of observation events in which mariner detects at least one flash pulse. More precisely, it is the probability the light produces an illuminance of at least 0.67 sea-mile-candela at the observer's position in any randomly selected contiguous time interval. Illuminance is determined from Allard's Law using effective intensity (EFI), distance, and a visibility of 10 nautical miles.

Probability of Recognition (POR). - The proportion of observation events in which a mariner detects all flash pulses within at least two adjacent flash periods. More precisely, it is the probability that in any randomly selected time interval, the light produces flash pulses for at least two contiguous flash periods which all have illuminances of at least 0.67 sea-mile-candela at the observer's position. Illuminance is determined from Allard's Law using effective intensity (EFI), distance, and a visibility of 10 nautical miles.

90% POD Distance. - The distance at which a mariner has a 90% probability of detecting a specified flashing light on a specified rolling buoy, while observing for a specified length of time.

90% POR Distance. - Same as 90% POD Distance except mariner recognizes characteristic according to definition of POR.

90% POD Nominal Range Factors. - A light's 90% POD Distance divided by its rated nominal range. For a particular lantern and flash characteristic, POD nominal range factor is a measure of range degradation of a buoy light due to buoy motion.

3.2 Assumptions and Conditions of Experiment

a. Buoy motion was recorded in only one plane. The mariner is located in this plane. Effects of buoy motion in this plane of observation are representative of effects of buoy motion in all other planes of observation.

b. The mariner knows where to look for the buoy. He looks continuously in the correct direction for a finite period of time. Thus if a flash produces at least 0.67 sea-mile-candela at the observer's position within the observation time, it will be detected.

c. The lamp flashes with instantaneous rise and fall times. Lamp filament nigrescence time effects are considered minimal compared to buoy motion effects. Thus instantaneous intensity in the mariner's direction is determined from buoy angle, lantern vertical beam profile, and whether or not the lamp is lit. The ideal square flash pulse is modified only by buoy roll and vertical divergence of the lantern.

d. The Schmidt-Clausen method (IALA (1978)) determines effective intensity (EFI) for each flash pulse. Flash pulse duration defines limits of integration for this method, not the actual instantaneous intensity in the horizontal plane.

e. If a mariner detects a flash pulse at a certain distance in a given observation time, he will detect the same pulse in greater observation time at that distance. He may also detect other flashes in the greater observation time.

f. If a mariner detects a flash pulse at a given distance within a certain observation time, he will also detect the same pulse at shorter distances within the same observation time. He may also detect other flash pulses at these shorter distances.

g. In order to recognize a flash characteristic, the mariner must detect all pulses within two contiguous flash periods. A detected flash pulse is considered a recognized flash pulse: apparent pulse width or fading in and out of a single pulse does not affect recognition. It is assumed a mariner can distinguish fading of a pulse due to buoy roll from electronic switching of a pulse. (Note that this is similar to distinguishing rotating beacons from electronically flashed lights.)

h. Visibility is assumed constant at 10 nautical miles (0.74 atmospheric transmissivity).

4. DATA ACQUISITION

Buoy angle was sampled every 0.2 seconds and recorded using an optical buoy motion recording system, shown in Figure 3. Light from the buoy was polarized in a fixed direction with respect to the buoy axis. As the buoy tilted, the plane of polarization rotated. The buoy motion recording system detected polarization rotation in a plane perpendicular to line of observation from the recording system. Software was developed (Appendix B - RECORD_BUOY_MOTION) to control the recording system and collect data.

A brief hardware description is as follows: The buoy motion recorder uses a rotating polaroid and phase sensitive detection to determine buoy rotation (or polarization rotation) angle. A Compumotor stepper motor (25,000 steps per revolution) rotates the polaroid at about 960 RPM. The rotating polaroid is located between a 600mm F/4.5 Canon lens and an EMI 9558 photomultiplier tube. A signal from an optical shaft encoder on the stepper motor locates a reference position of the polaroid. The shaft encoder signal, after passing through a divide-by-N counter, provides two pulses per revolution (32 Hz) of the polaroid. Polarized light incident on the lens results in a sinusoidal signal from the photomultiplier tube with a DC offset. The DC

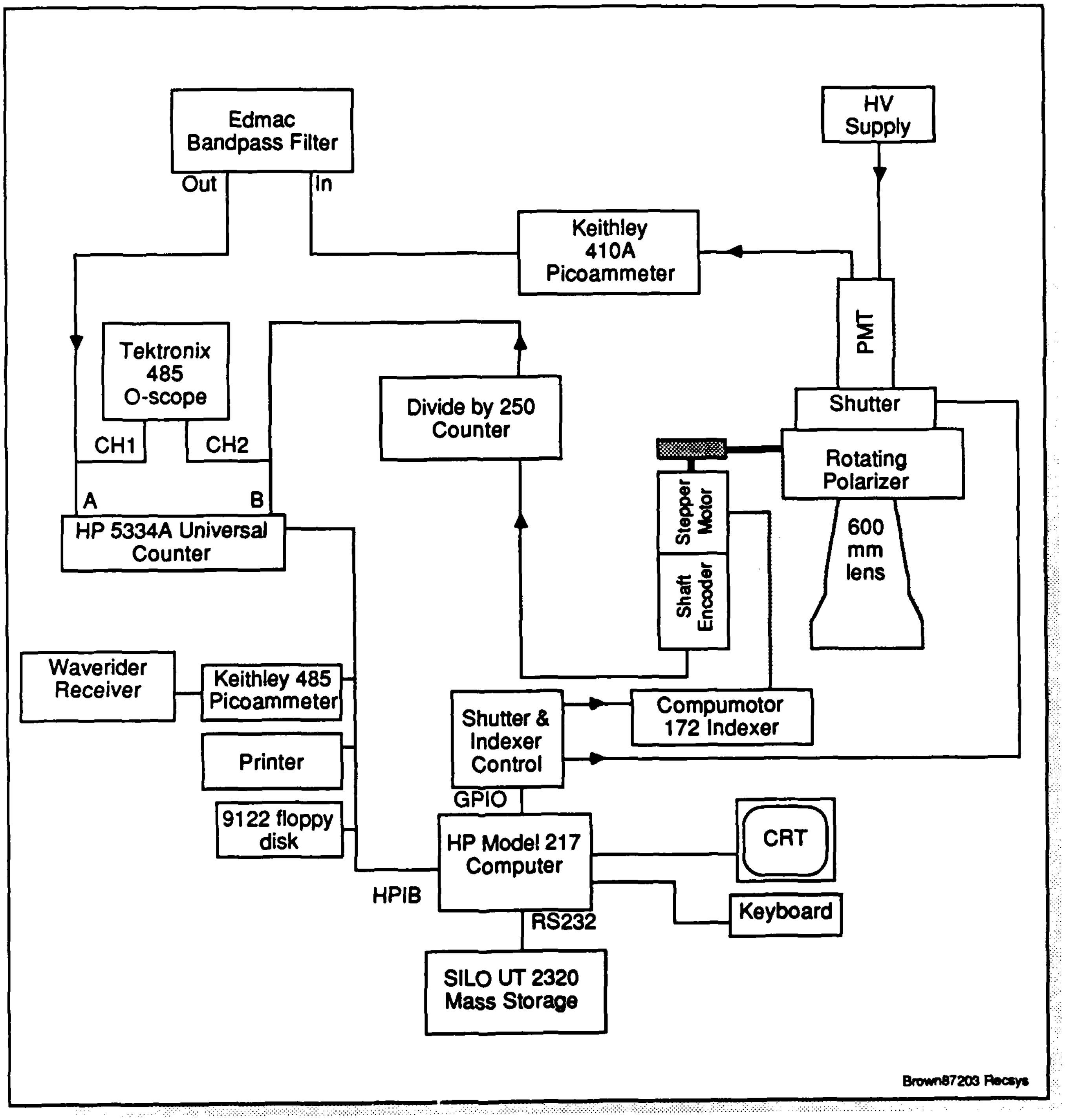


FIGURE 3. Optical Buoy Motion Recording System

offset is due to unpolarized light. An Edmac bandpass filter removes noise and DC level from detector signal. An HP 5334 universal counter determines phase between shaft encoder and photomultiplier signals. Phase is sampled by the computer every 0.2 seconds and converted to instantaneous buoy angle.

Laboratory measurements showed the system to have a detection threshold of 0.73 sea-mile-candela. Field tests showed atmospheric scintillation and polarized moonlight reflected off the water significantly reduced sensitivity. Atmospheric scintillation was partially overcome by extending the source so that light arrived at the lens over multiple paths. Data collection was avoided during moonlit nights.

Two buoys, an 8x26 and 5x11 (see Figure 4), were moored off Avery Point, Groton, CT to provide actual buoy motion. The 8x26 buoy was installed with a battery powered polarized light source. A 30-inch fluorescent lamp worked best for overcoming atmospheric scintillation. A 2-foot cylinder of retroreflective sheeting was wrapped with polaroid sheeting and installed on the 5x11 buoy (see Figure 5). Retroreflective material served to reflect light from a 1000-watt xenon arc searchlight to the detector.

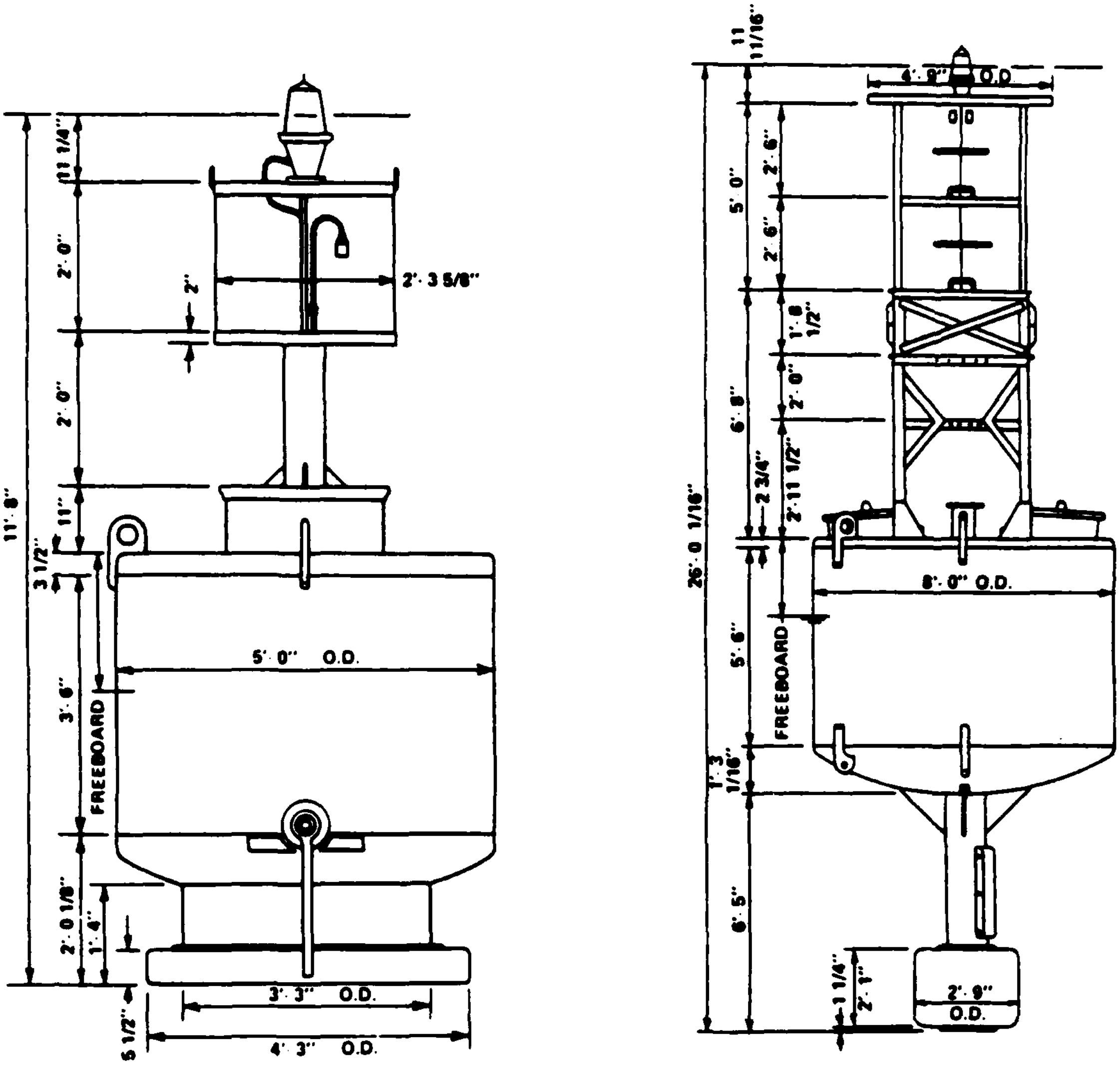
Five 30-minute continuous recordings of buoy roll angle versus time were obtained on two buoys. More recordings, spanning a larger range of conditions, were planned; time, atmospheric conditions, and hardware failures limited data collection. These recordings are listed in *TABLE* 1.

TABLE 1

BUOY DATA RECORDINGS

DATE	BUOY	RMS ROLL	WIND	WAVE HT (est)
12/4/86	8x26	3.3 deg	N/A	N/A
11/13/86	8x26	8.0 deg	20-25 kts	2 - 3 ft
11/13/86	8x26	9.9 deg	20-25 kts	2 - 3 ft
2/13/87	5x11	8.2 deg	5 -10 kts	1 - 2 ft
2/13/87	5x11	8.4 deg	10-20 kts	1 - 2 ft

No effort was made to quantify the frequency distribution of sea states. Long term daily monitoring would be required to collect this information. It is the author's opinion that conditions experienced during the period of data collection are typical of exposed and semi-exposed locations.



A) 5 X 11 BUOY
WEIGHT 3004 LBS

B) 8 X 26 BUOY
WEIGHT 11,382 LBS

FIGURE 4. Subject Buoy Dimensions and Weights

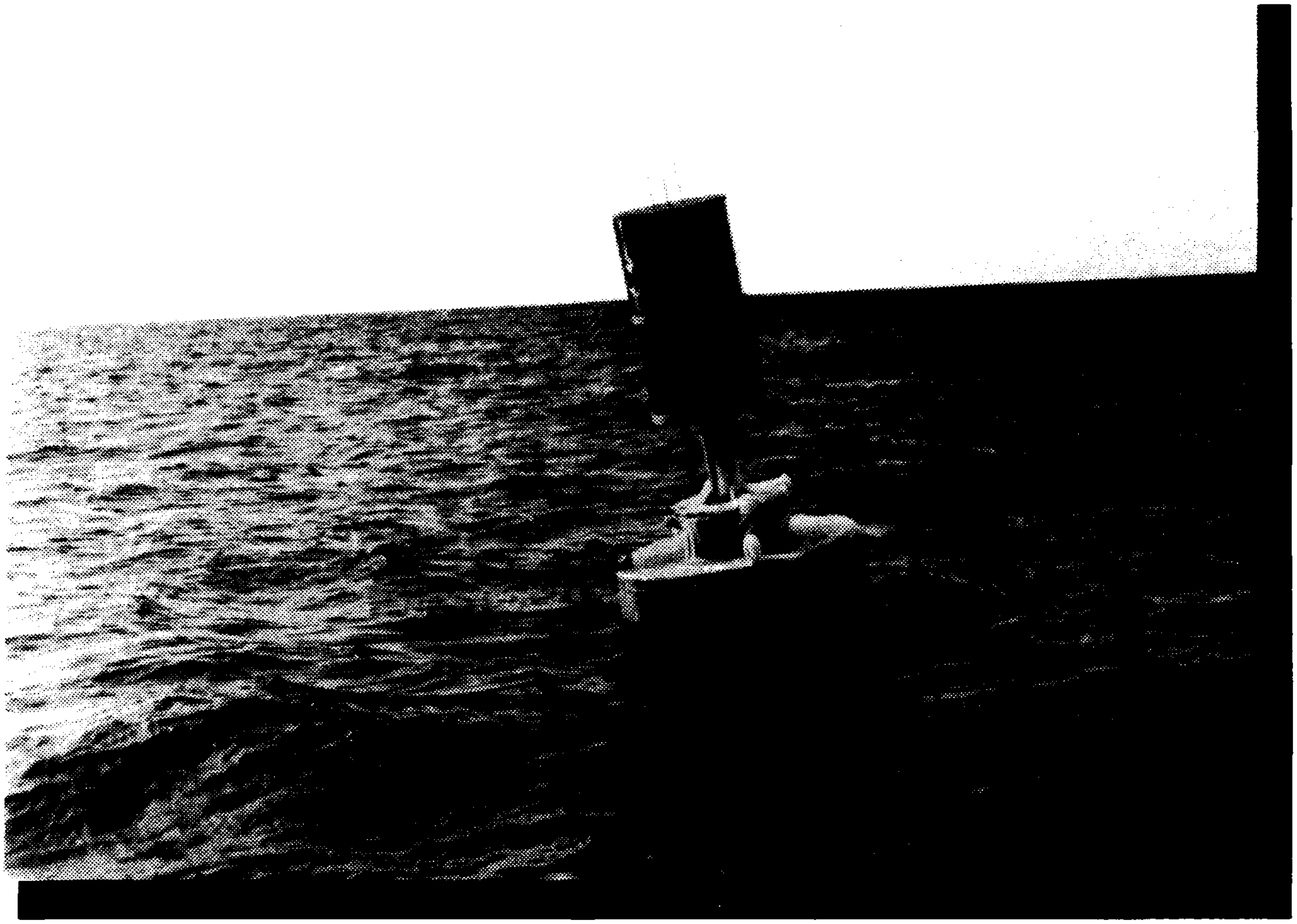


FIGURE 5. 5 X 11 Buoy with Polarizing Retroreflector

5. DATA ANALYSIS

5.1 Probability Calculations

Software was developed to calculate POD and POR as a function of observation time, observation distance, flash characteristic, lens vertical divergence, and buoy motion (Appendix B - BUOY_PROBABILITIES). The program required two data file inputs, a recording of buoy angle versus time and vertical intensity profile data for the lantern. Vertical intensity profile was provided in 0.2-degree increments. It was used to convert buoy roll angle to intensity in the horizontal plane. The ten standard flash characteristics listed in USCG (1983) were used in the analysis. These characteristics are defined in Figure 6 (taken from USCG (1983)).

Table 2 lists lanterns, with respective file names, used to determine detection and recognition probabilities. File names are given for convenience in identifying the data. The 155mm and 200mm lantern profiles were measured in the photometric laboratory at the U.S. Coast Guard Research and Development Center. The last five lantern profiles are simulated profiles. These gaussian profiles provide the same total flux as the yellow 155mm lantern. Software was written to generate these profiles (Appendix B - LanternProfileMaker) to study the effect of vertical divergence on detection and recognition probabilities.

TABLE 2

LANTERN DATA FILES

FILE NAME	LANTERN	LAMP	FWHM
LANT	Yellow 155mm	1.15A	4.2 deg
LAN155_55	Clear 155mm	.55A	4.3 deg
LAN155_115	Clear 155mm	1.15A	4.2 deg
LAN200_115	Clear 200mm	1.15A	4.0 deg
GAU4	Yellow Gaussian	1.15A	4.2 deg
GAU5	Yellow Gaussian	1.15A	5.0 deg
GAU10	Yellow Gaussian	1.15A	10.0 deg
GAU15	Yellow Gaussian	1.15A	15.0 deg
GAU20	Yellow Gaussian	1.15A	20.0 deg

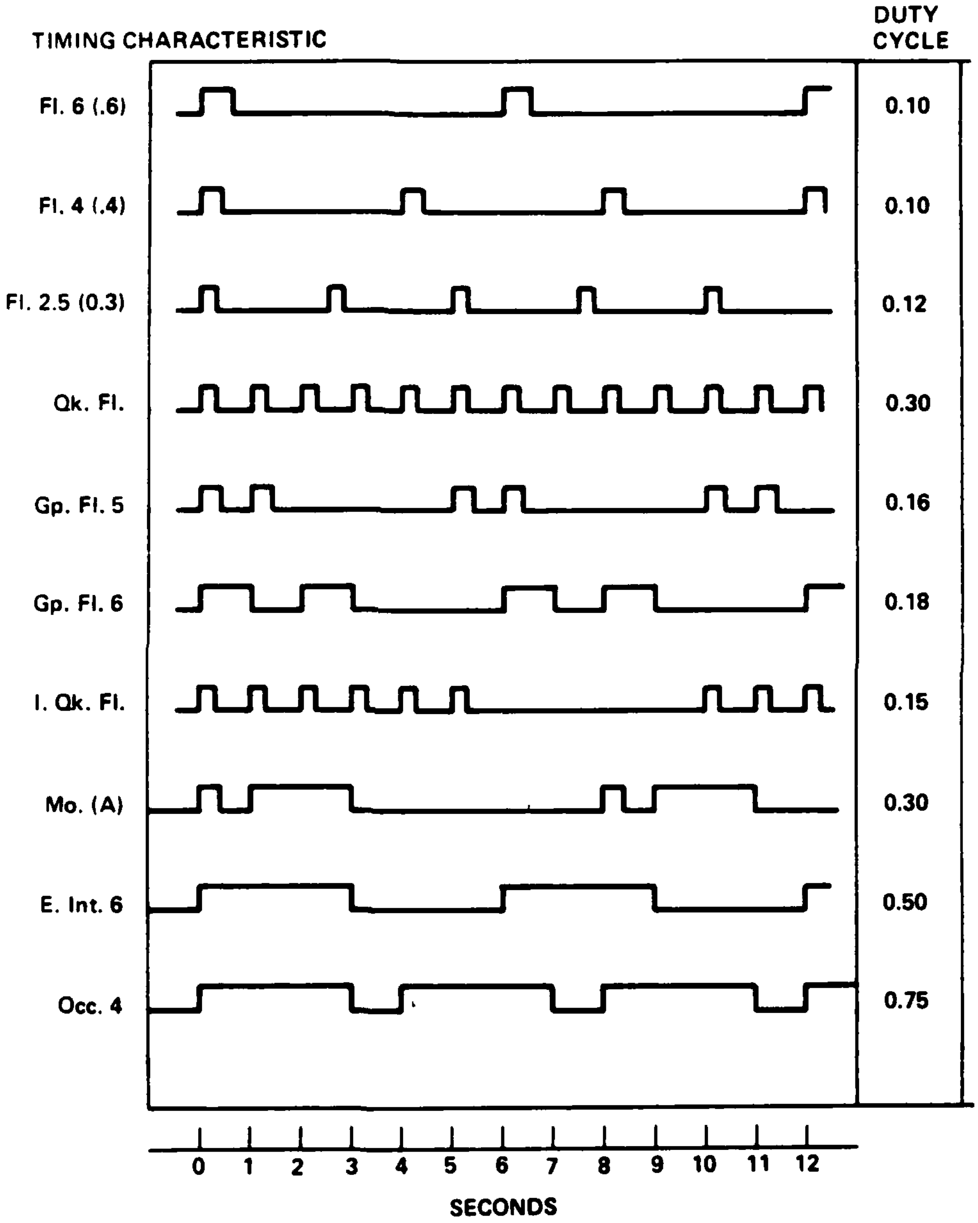


FIGURE 6. Standard Flash Characteristics

Buoy angle versus time was converted to intensity as a function of time in the horizontal plane for a fixed-on light. This was done by mapping buoy roll angles into the appropriate vertical divergence file. Prior to mapping, additional buoy angle data points were interpolated to increase intensity resolution. Interpolation gave an effective intensity sampling rate of 35 Hz.

A normalized flash characteristic was then superimposed onto the intensity versus time function. Five hundred 30-second windows were randomly selected from the fixed-on light intensity versus time record and multiplied by a normalized flash characteristic. Phase between selected windows and normalized flash characteristic was also randomized. Software testing with generated sinusoidal and cosinusoidal buoy roll data assured proper randomization in the algorithm.

Effective intensities were calculated (using the Schmidt-Clausen method) for each flash pulse in the 30-second window. In each window, time periods (corresponding to observation times) varying from 2 to 30 seconds were searched for minimum and maximum intensities. Minimum intensities determined POR and maximum intensities determined POD for various combinations of observation distance and observation time.

Figure 7 illustrates the procedure for converting buoy roll versus time to flashed intensity versus time for one 30-second data window. The top graph is a typical 30-second recording of buoy angle versus time. The middle graph is this recording converted to intensity versus time for a fixed-on light. The bottom graph is the product of the middle graph and a normalized FL 6 (0.6) flash characteristic. For this particular window, the observer (depending on distance) might detect the first two flash pulses but not the others. This window would allow detection for all observation times from 2 seconds to 30 seconds. Had these two pulses occurred at 24 and 30 seconds, detection would not occur until 24 seconds of observation time had elapsed.

Matrices of detection and recognition probabilities as functions of observation time and distance were calculated for each of ten standard flash characteristics and for various combinations of buoy roll data and lantern types. Results presented here are extracted from a collection of 280 such matrices, far too many to be reproduced in their entirety[1]. Figures 8 and 9 are examples of these matrices. Figure 8 shows POD of a perfectly vertical light (zero degree roll amplitude). The light, a 155 mm lantern, has a peak intensity of 232 candela. With a 0.6-second contact closure time, this light produces a nominal range of 6 nautical miles according to USCG (1984). Indeed, Figure 8 shows this is a

[1] The 280 data matrices can be obtained by writing to Commanding Officer, USCG Research and Development Center, Avery Point, Groton, CT 06340-6096, Attention: Chief, Physics Branch.

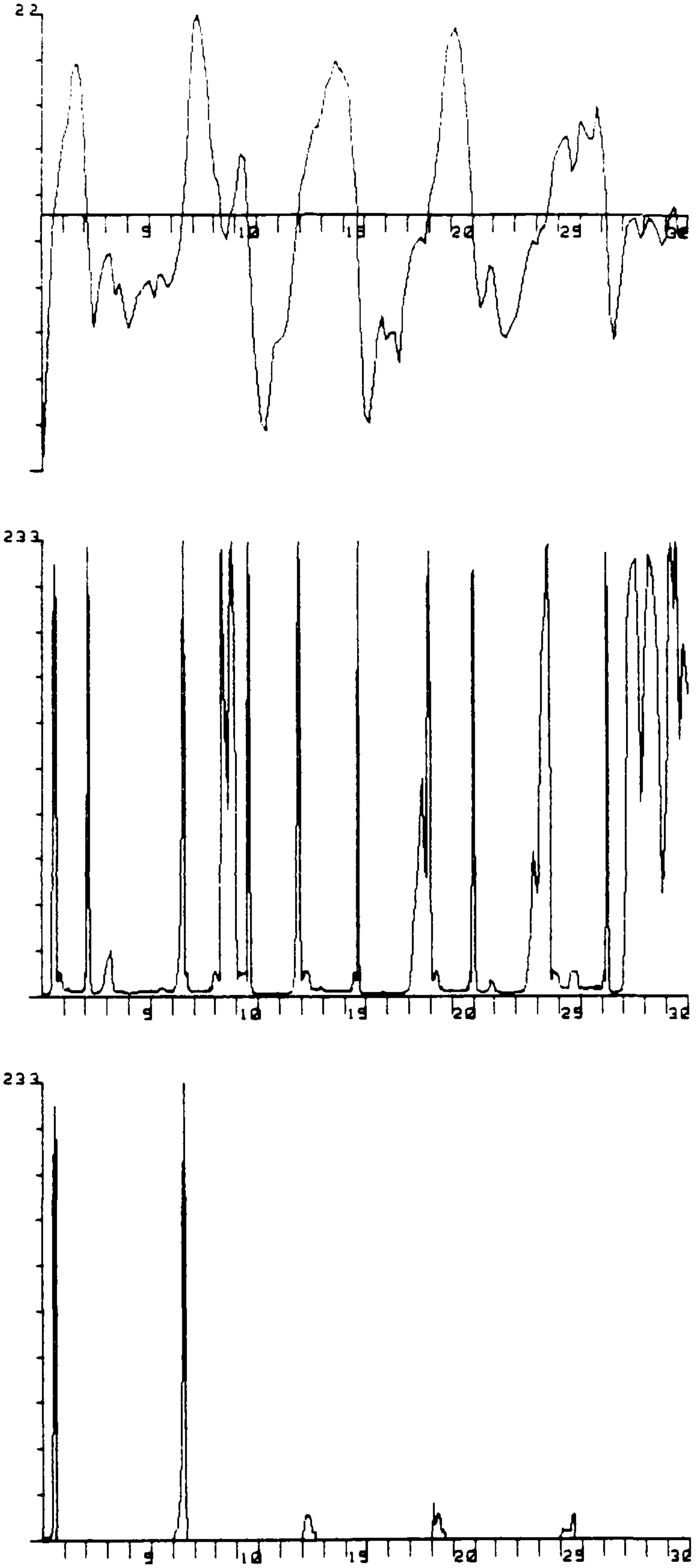


FIGURE 7. Buoy Angle To Flashed Intensity Conversion

Probability of Detection
FL. 6 <0.6>

Distance	Observation Time (s)														
Nt.Mi.	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
0.5	30%	67%	99%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
1.0	30%	67%	99%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
1.5	30%	67%	99%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
2.0	30%	67%	99%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
2.5	30%	67%	99%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
3.0	30%	67%	99%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
3.5	30%	67%	99%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
4.0	30%	66%	99%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
4.5	29%	66%	98%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5.0	29%	66%	98%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5.5	29%	65%	98%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
6.0	25%	62%	94%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
6.5	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
7.0	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
7.5	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
8.0	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%

FIGURE 8. Typical POD Matrix For Zero Roll

Probability of Detection
FL. 6 (0.6)

Distance	Observation Time (s)														
Nt.Mi.	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
0.5	30%	67%	99%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
1.0	29%	64%	96%	98%	98%	99%	99%	100%	100%	100%	100%	100%	100%	100%	100%
1.5	22%	52%	78%	84%	88%	92%	94%	95%	97%	90%	99%	99%	99%	99%	99%
2.0	21%	47%	70%	76%	82%	88%	91%	92%	95%	95%	97%	98%	98%	98%	99%
2.5	18%	40%	58%	64%	72%	79%	83%	85%	89%	90%	93%	94%	94%	95%	96%
3.0	17%	35%	52%	59%	68%	75%	78%	81%	86%	88%	91%	92%	92%	94%	95%
3.5	16%	33%	49%	56%	65%	73%	77%	79%	84%	86%	89%	91%	92%	94%	95%
4.0	15%	31%	47%	54%	63%	70%	74%	77%	82%	84%	88%	90%	91%	93%	94%
4.5	14%	28%	42%	49%	59%	65%	69%	72%	78%	81%	85%	87%	87%	90%	92%
5.0	11%	25%	38%	45%	54%	61%	65%	68%	73%	75%	79%	82%	83%	86%	88%
5.5	6%	15%	25%	31%	39%	44%	48%	52%	57%	60%	63%	65%	68%	70%	73%
6.0	2%	4%	6%	8%	11%	13%	15%	18%	20%	22%	24%	25%	26%	28%	30%
6.5	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
7.0	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
7.5	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
8.0	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%

FIGURE 9. Typical POD Matrix For Non-zero Roll

6-mile light. Figure 9 shows the probability of detecting the same light with only 10 degrees rms buoy roll. It is now something less than a 6-mile light.

5.2 Frequency Spectra of Buoy Roll

Since detection and recognition probabilities depend on buoy roll, it is desirable to characterize the rolling motion of a buoy. Spectral analysis is one tool for characterizing motion. A frequency spectra or power spectral density (PSD) gives the roll amplitude (or energy) at various frequencies. The usefulness of this tool is demonstrated by the fact that although waves and buoys move semi-randomly, their power spectral densities are constant over long periods of time.

Additionally, if a buoy is assumed to be a linear system, its PSD may be determined from wave PSD by a simple multiplication of two frequency curves. If sea spectra were flat (white noise), buoy PSD peaks would correspond exactly to a buoy's natural roll frequency. But sea spectra are not flat. They contain predominant frequencies which produce actual roll frequencies slightly different from natural roll frequencies. Fortunately, researchers have measured wave PSD in waterways all over the globe. Thus one should be able to predict the roll spectra of a buoy for practically anywhere in the world, given the mechanical response of the buoy to waves (called the transfer function) is known.

This is analogous to predicting the output of an electrical network or filter for random noise input. The time interval over which the output signal of a filter is correlated to a single frequency approximately equals the reciprocal of its full bandpass. Likewise for a buoy, the time interval over which a buoy rolls at a constant frequency is indicated by the width of roll PSD peaks. This correlation time was discovered to affect POD and POR, thus the importance of buoy roll spectral analysis. These results are discussed in section 6.5.

Efforts to measure buoy transfer functions directly and verify response linearity were thwarted by failure of the wave-rider buoy. It was not discovered until late in the experiment that marginal operation of the wave-rider buoy produced invalid wave height data. However, responses to step inputs on both buoys were measured. This step input is used to determine the natural roll frequency of the buoy, as well as the buoy transfer functions and frequency response. Appendix A discusses the mathematics of spectral analysis, the linearity assumption, and derivation of transfer functions from step responses[2].

[2] Software for making these calculations can be found in Appendix B - FastFourierTransform.

A typical step response recording for the 5x11 buoy is shown in Figure 10. Figures 11 and 12 show relative magnitudes of transfer functions derived from 5x11 and 8x26 step responses according to Appendix A. Figure 11 shows the natural roll frequency of the 5x11 buoy to be 0.273 Hz, giving a roll period of 3.7 seconds. Figure 12 shows the natural roll frequency of the 8x26 buoy to be 0.156 Hz, giving a roll period of 6.4 seconds. The 8x26 buoy has a narrower bandpass of frequencies, meaning less damping and longer correlation times.

6. RESULTS

6.1 Observation Time and Distance Effects

As expected POD and POR are strongly affected by observation distance and length of time spent looking in the direction of the buoy. Figures 13 and 14 show the relationship of POD and POR to observation time at a position of 4 nautical miles from an 8x26 buoy with 10 degrees rms roll. POD approaches maximum faster for high flash rate characteristics than for low flash rate characteristics.

An unexpected result shows POR is higher for FL 2.5 than for QK FL. This result is found consistently with other rms amplitudes and with the 5x11 buoy. This is due to correlation between flash period and buoy roll period and is discussed in section 6.5

Figures 15 and 16 show the relationship of POD and POR to observation distance at a constant 12-second observation time. These data are from the 8x26 buoy with 10 degrees rms roll. As expected POD drops with distance from the buoy and drops more slowly for the higher flash rate characteristics. Again POD is higher for the QK FL, but POR is higher for the FL 2.5.

6.2 Performance Measurement

Variation of POD and POR with observation time and distance makes it difficult to measure, specify, or compare lantern performance. Comparing POD and POR performance of different buoys, lanterns and flash characteristics is easier if the probability matrices are reduced to a single number, similar to currently used nominal range. This was done by searching the matrices for the distance at which 90% POD occurs with a 12-second observation time. Since higher PODs occur at shorter distances (or longer observation times) than this 90% POD distance, it is a reasonable measure of system performance. A 90% POD level and 12-second observation time were arbitrarily chosen. (See section 3.1 definitions for 90% POD and POR distances.)

Ninety percent (90%) POD and POR distances for 12-second observation were determined from the probability matrices and are given as Tables 3 and 4. These data are for a yellow 155mm lantern with a 1.15-ampere lamp. Distances are given in nautical

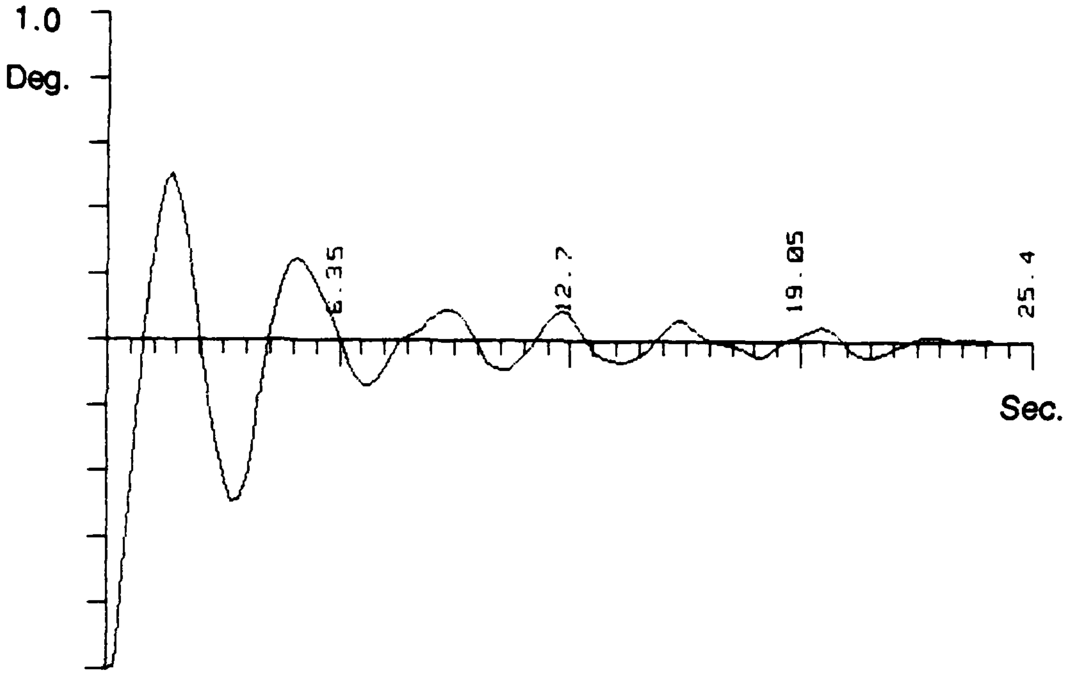


FIGURE 10. Typical Step Response for 5 X 11 Buoy

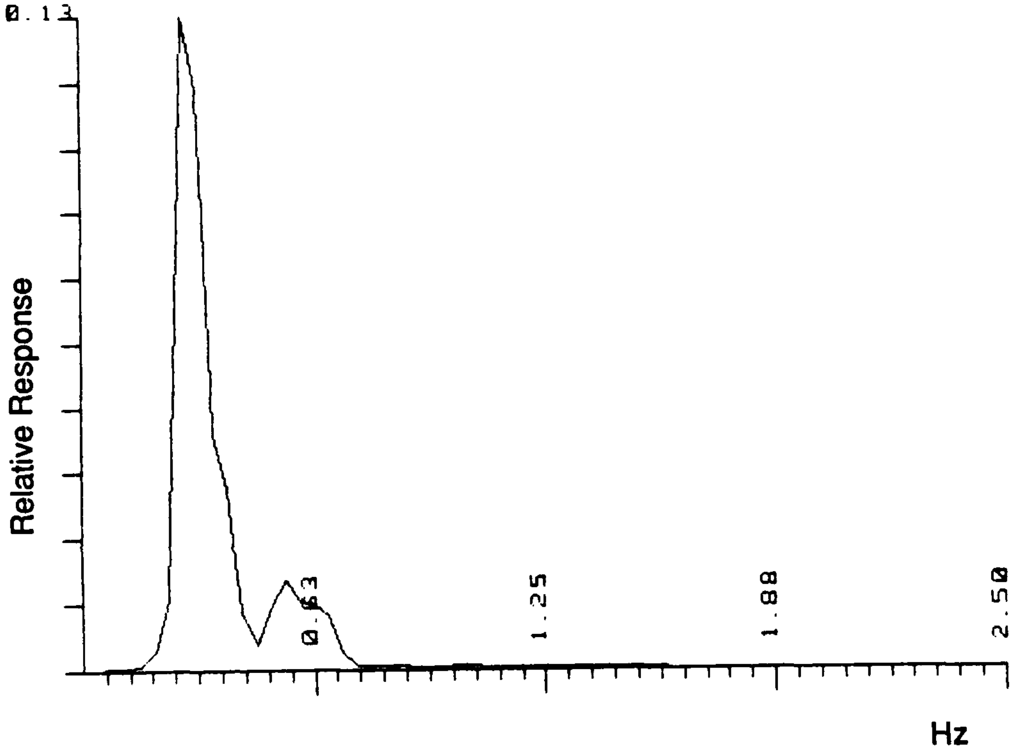


FIGURE 11. 5 x 11 Buoy Step Response RAO

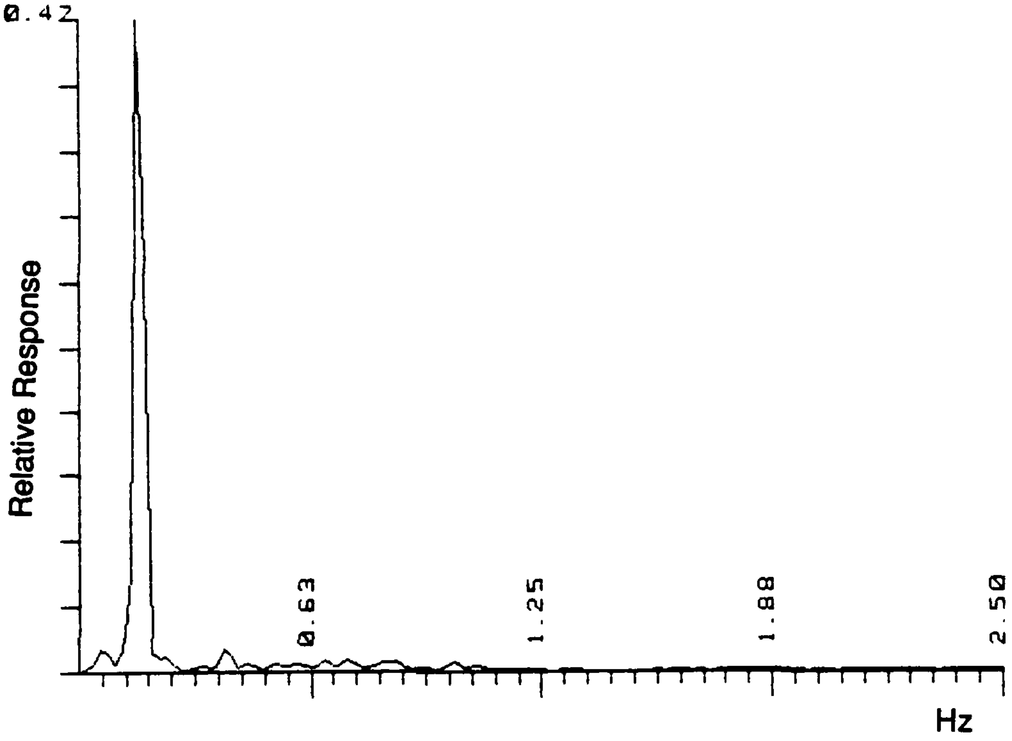


FIGURE 12. 8 x 26 Buoy Step Response RAO

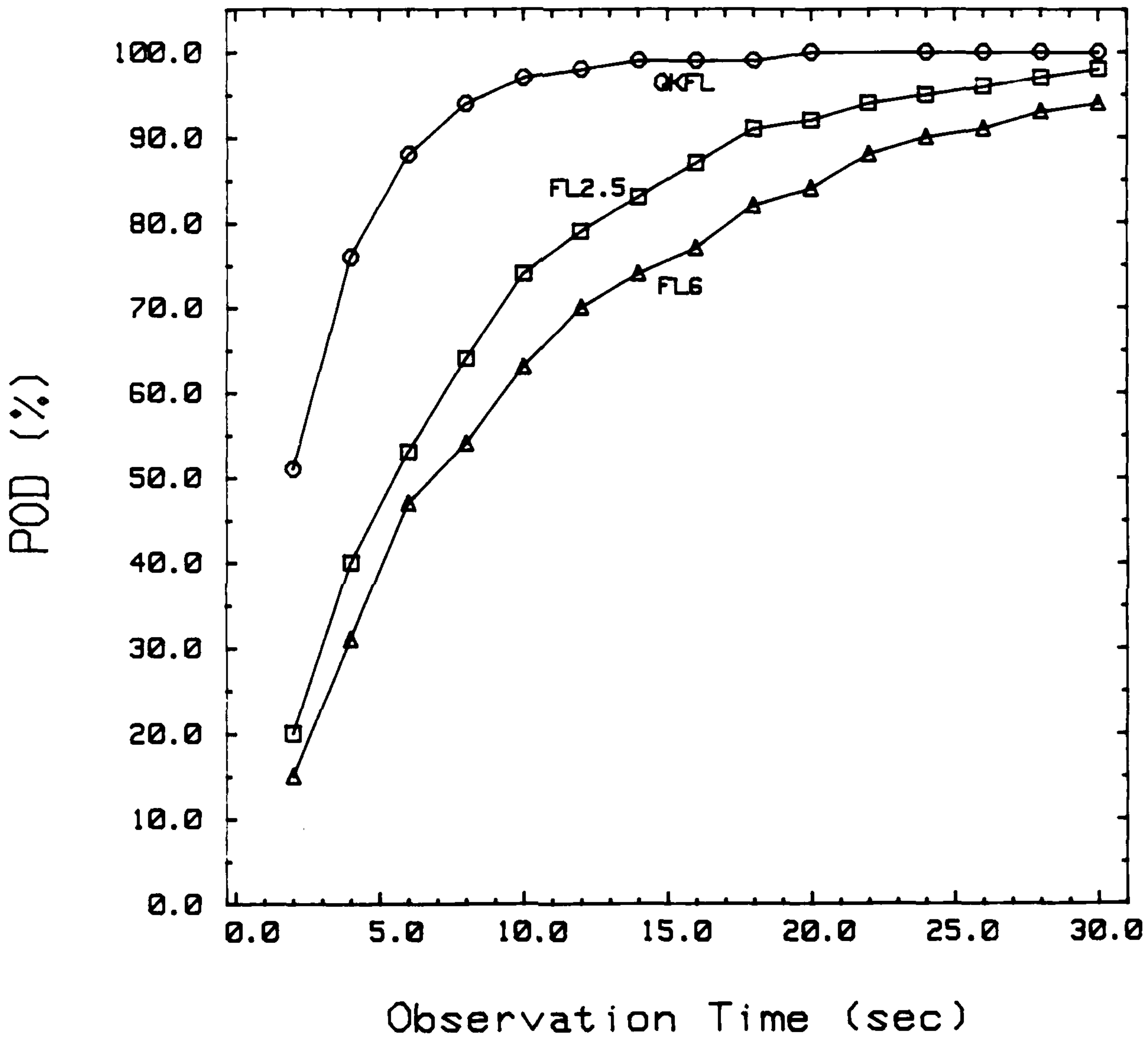


FIGURE 13. POD Versus Observation Time at 4 Nautical Miles

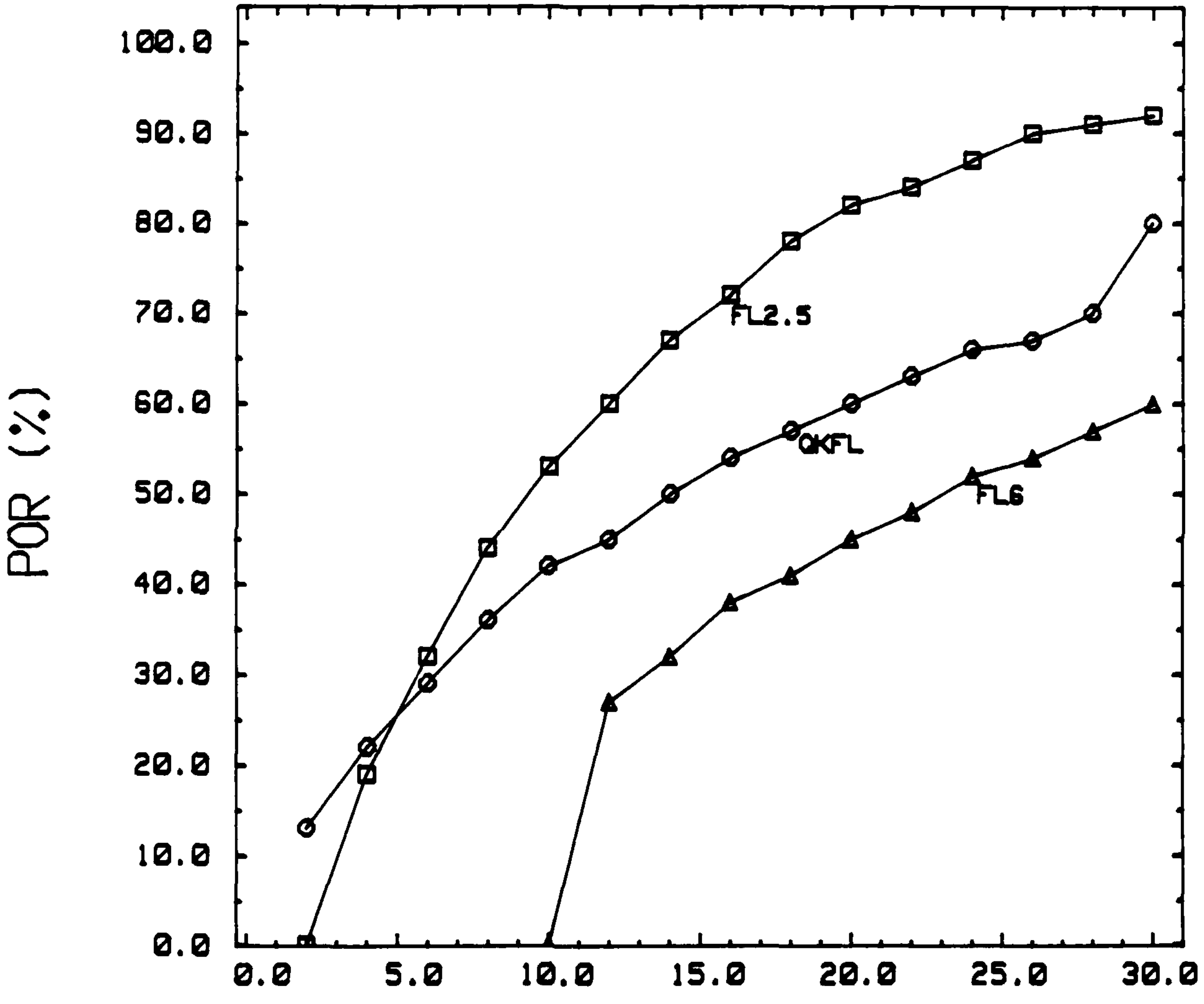


Figure 14. POR Versus Observation Time At 4 Nautical Miles

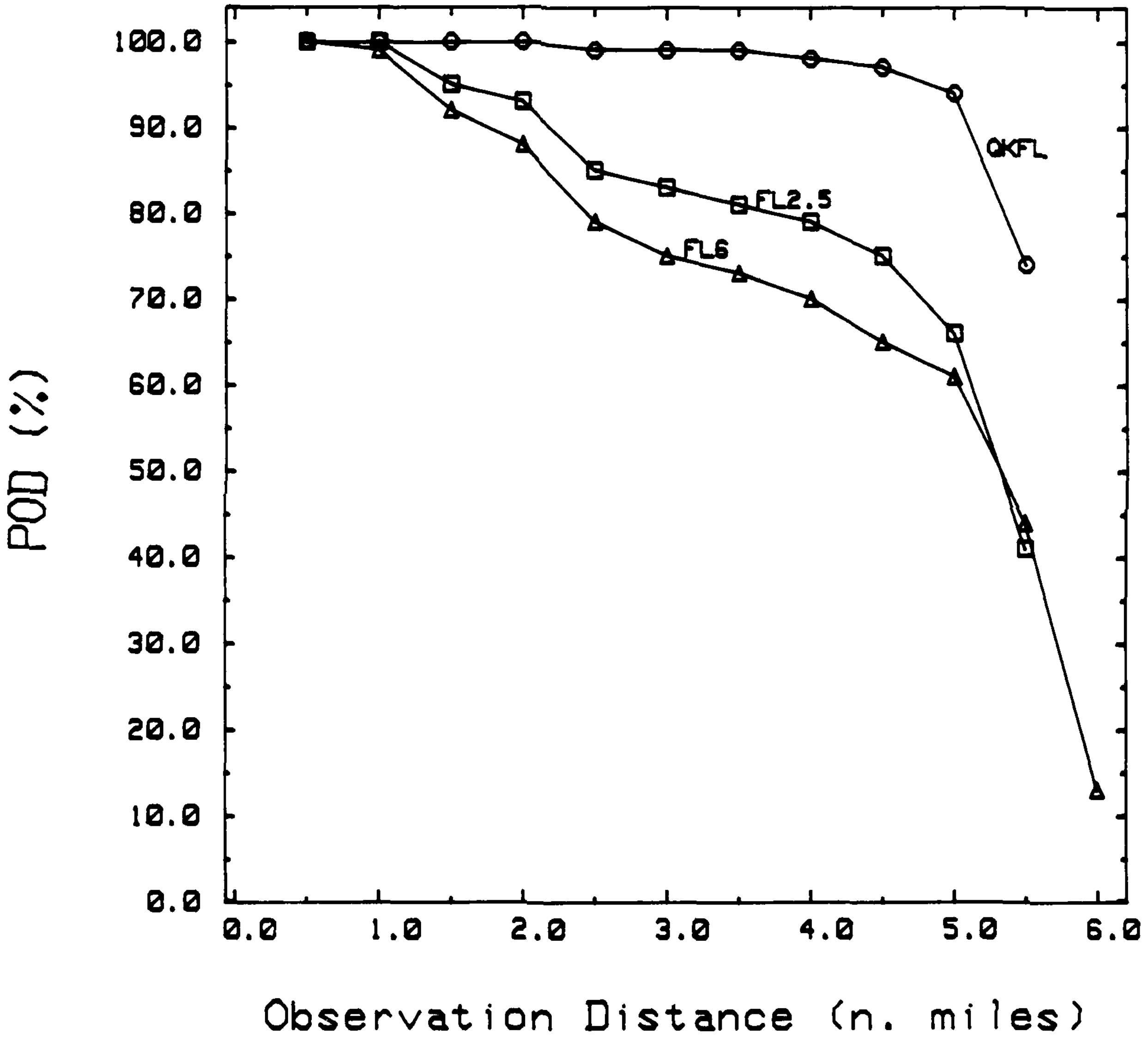


FIGURE 15. POD Versus Observation Distance With 12-Second Observation Time

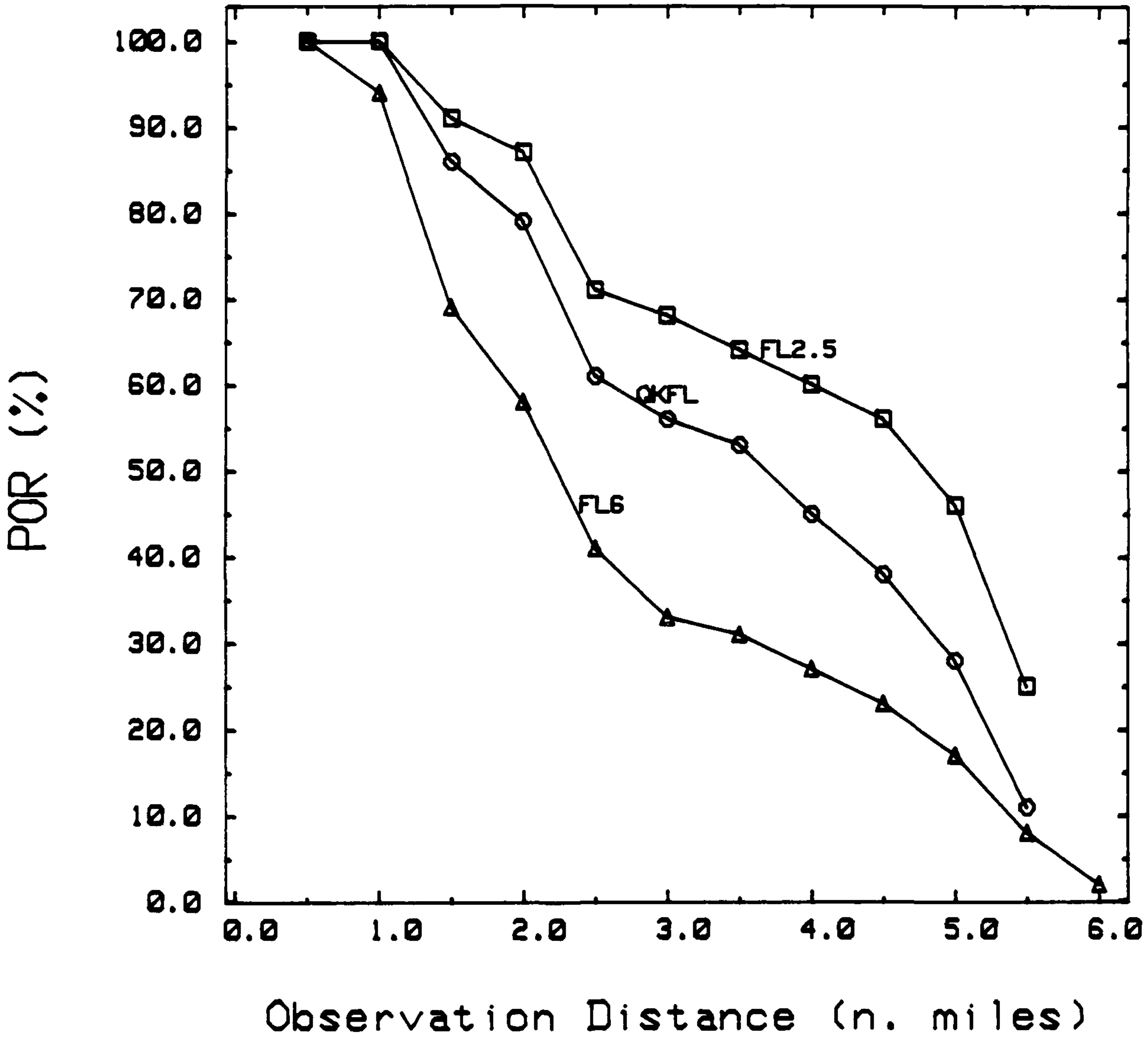


FIGURE 16. POR Versus Observation Distance With 12-Second Observation Time

miles as a function of buoy roll and flash characteristic. Column labels give buoy type / rms roll amplitude (degrees). Row labels give flash characteristic. Ninety percent (90%) POR distances are not available for the I QK FL and MO (A) characteristics since two periods of these characteristics exceeds the 12-second observation time.

It is apparent that probabilities of detecting and recognizing flashing lights on rolling buoys are a strong function of flash characteristic and buoy motion. Characteristics with low duty cycles and low repetition rates have lowest probabilities of detection and recognition. Characteristics with high repetition rates or high duty cycles have highest probabilities of detection and recognition.

TABLE 3

90% POD DISTANCES FOR 12-SECOND OBSERVATION VERSUS BUOY MOTION

CHAR.	N.R.*	8x26/3°	8x26/8°	8x26/10°	5x11/8.2°	5x11/8.4°
FL 6(.6)	6.3	2.8	1.8	1.8	2.1	2.1
FL 4(.4)	6.1	5.1	2.2	2.2	2.1	2.1
FL 2.5(.3)	5.9	5.2	2.3	2.2	2.8	3.0
QK FL	5.9	5.5	5.1	5.1	5.1	5.3
GP FL 5	6.1	5.5	3.0	3.0	3.8	3.8
GP FL 6	6.4	5.9	5.3	5.1	5.3	5.1
I QK FL	5.9	5.5	4.5	4.5	4.4	4.6
MO (A)	6.5	6.0	5.2	5.2	5.5	5.5
E INT 6	6.6	6.1	5.7	5.6	5.8	5.7
OCC 4	6.6	6.2	5.9	5.8	6.0	5.8

*N.R. = Nominal Range determined for lamp/lantern combination measured in photometric laboratory.

Ninety percent (90%) POD distances for each light may be compared with its nominal range rating by calculating 90% POD Nominal Range Factors, the ratio of 90% POD distance to nominal range. Nominal Range Factors for the yellow 155mm lantern are provided by Table 5. Comparing flash characteristics of identical duty cycle (e.g. FL 6 versus FL 4, QK FL versus MO (A)) indicates that repetition rate is more important than duty cycle for preventing nominal range degradation. As would be expected, QK FL produces the highest nominal range factors and FL 6 produces the lowest.

It is clear that nominal range is not a good measure of buoy lantern performance. Nominal range may be more than 300% in error. For example, an FL6 (0.6) characteristic provides a greater nominal range (due to its longer contact closure time) than the QK FL characteristic (US Coast Guard; 1984), but results reported here showed that QK FL performed significantly better than FL6 (0.6). In some cases QK FL produced more the 3 times the 90% POD distance as FL6 (0.6).

TABLE 4

90% POR DISTANCES FOR 12-SECOND OBSERVATION VERSUS BUOY MOTION

FLASH CHAR.	8x26/3°	8x26/8°	8x26/10°	5x11/8.2°	5x11/8.4°
FL 6(.6)	2.2	1.1	1.1	1.2	1.2
FL 4(.4)	2.1	1.1	1.1	1.2	1.1
FL 2.5(.3)	3.8	1.9	1.6	2.0	2.1
QK FL	3.0	1.7	1.4	1.5	1.6
GP FL 5	2.0	0.8	0.8	1.0	1.0
GP FL 6	2.2	1.1	1.1	1.2	1.2
I QK FL	N/A	N/A	N/A	N/A	N/A
MO (A)	N/A	N/A	N/A	N/A	N/A
E INT 6	4.7	2.5	2.5	3.5	2.6
OCC 4	6.1	5.4	5.3	5.6	5.5

Higher POD does not always imply higher POR, as shown by comparing the two group flash characteristics (GP FLs) with FL 2.5 in Tables 3 and 4. Both GP FL 5 and GP FL 6 have higher POD distances than FL 2.5 because of longer flash durations, but lower POR distances due to longer flash periods. A similar anomaly occurs with QK FL and FL 2.5 due to correlation between flash period and roll period (section 6.5).

6.3 Vertical Divergence Effects

As vertical divergence increases, optical plane intensity (peak intensity) decreases, but the chance that a light will be visible through the duration of a pulse increases. Effects of increasing vertical divergence were investigated by calculating detection and recognition probabilities for lanterns with various vertical divergence profiles. Vertical divergence is represented by the full vertical width of the beam at half peak intensity. Profiles were simulated by distributing light in a Gaussian form with full divergence widths of 4.2, 5, 10, 15, and 20 degrees (see Table 2). All simulated profiles contained the same total light flux, that of a yellow 155mm / 1.15A lantern. Calculations show the effects of changing the lens, but maintaining the same lamp and power consumption.

Buoy roll data were taken from the 8x26 buoy with 10 degrees rms roll. Ninety percent (90%) POD distances were determined from probability of detection matrices. Table 6 gives 90% POD distances for these Gaussian profiles. 90% POD distances for the yellow 155mm are also repeated for convenience under "LANT". These data are plotted in Figure 17 for four different characteristics.

Increasing vertical divergence increases detection distance for flash characteristics with low repetition rates and low duty cycles, but slightly degrades detection distance for characteristics with high repetition rates or high duty cycles. The FL 6 characteristic benefits most from increasing vertical divergence. The OCC 4 characteristic benefits least from increasing vertical divergence.

TABLE 5

90% POD NOMINAL RANGE FACTORS FOR 12-SECOND OBSERVATION VERSUS BUOY MOTION

FLASH CHAR.	8x26/3°	8x26/8°	8x26/10°	5x11/8.2°	5x11/8.4°
FL 6(.6)	.44	.29	.29	.33	.33
FL 4(.4)	.84	.36	.36	.34	.34
FL 2.5(.3)	.88	.39	.37	.47	.51
QK FL	.93	.86	.86	.86	.90
GP FL 5	.90	.49	.49	.62	.62
GP FL 6	.92	.83	.80	.83	.80
I QK FL	.93	.76	.76	.75	.78
MO (A)	.92	.80	.80	.85	.85
E INT 6	.92	.86	.85	.88	.86
OCC 4	.94	.89	.88	.91	.88

A figure of merit for each profile can be obtained by calculating 90% POD distances weighted by frequency of occurrence of each characteristic (Table 6). A random sampling of 62 lighted buoys from a current Coast Guard Light List gave the following

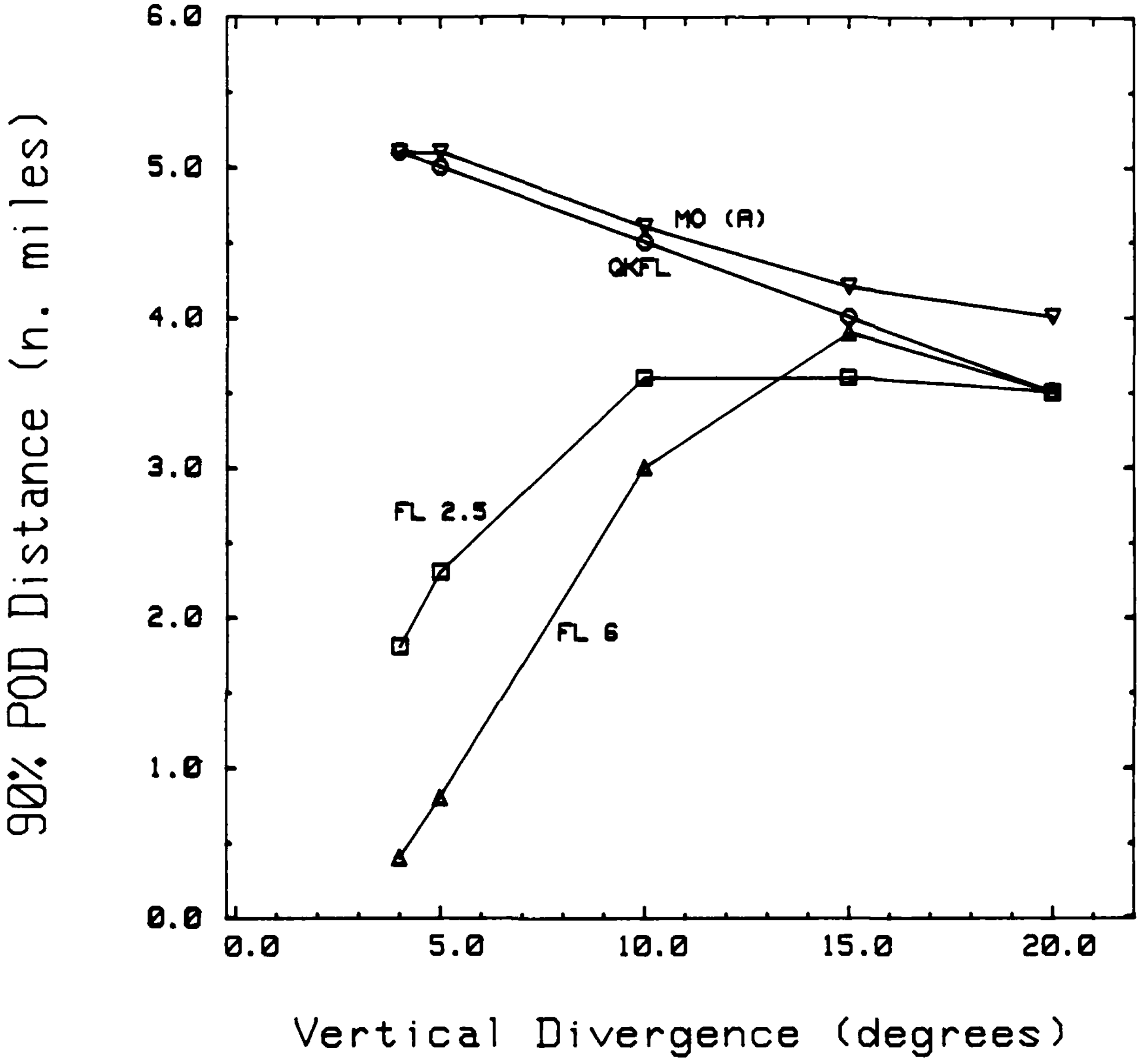


FIGURE 17. 90% POD Distance Versus Vertical Divergence For 12-Second Observation Time

distribution of flash characteristics:

FL 6 (.6)	4	6%
FL 4 (.4)	43	69%
FL 2.5 (.3)	5	8%
QK FL	5	8%
MO (A)	5	8%

Applying these weights gave weighted average 90% POD distances listed in Table 6. These averages predict that a lens with 10 to 15 degrees vertical divergence is optimum. This lens would yield a 73% average increase in 90% POD distance. In the case of the most commonly used characteristic, FL 4(.4), doubling the divergence nearly doubles the detection range.

TABLE 6

90% POD DISTANCES FOR 12-SECOND OBSERVATION VERSUS LANTERN DIVERGENCE

FLASH CHAR.	LANT	GAU4	GAU5	GAU10	GAU15	GAU20
FL 6(.6)	1.8	0.4	0.8	3.0	3.9	3.5
FL 4(.4)	2.2	1.8	2.5	3.7	3.7	3.5
FL 2.5(.3)	2.2	1.8	2.3	3.6	3.6	3.5
QK FL	5.1	5.1	5.0	4.5	4.0	3.5
GP FL 5	3.0	3.5	4.0	4.1	4.0	3.5
GP FL 6	5.1	5.1	5.1	4.6	4.1	4.0
I QK FL	4.5	4.5	4.5	4.2	4.0	3.5
MO (A)	5.2	5.1	5.1	4.6	4.2	4.0
E INT 6	5.6	5.6	5.5	5.0	4.5	4.0
OCC 4	5.8	5.8	5.6	5.0	4.5	4.0
Wt. Average	2.7	2.2	2.8	3.8	3.8	3.5

In order to resolve the long standing 200mm versus 155mm lantern debate among Coast Guard engineers, POD calculations were made using vertical profiles of these two lanterns. The profiles were measured in the USCG R&D Center photometric laboratory using identical lamps. Weighted averages were calculated as above and are given in Table 7. Some characteristics are slightly better on the 200mm lantern and others are better on the 155mm lantern. Results show the 200mm lantern provides only 3% more average range than the 155mm lantern. Clearly, there is very little difference between the two lanterns.

TABLE 7

COMPARISON OF 155mm AND 200mm CLEAR LANTERNS
(90% POD DISTANCES)

FLASH CHAR.	200mm/1.15A	155mm/1.15A
FL 6(.6)	2.4	1.9
FL 4(.4)	2.9	2.6
FL 2.5(.3)	2.7	2.5
QK FL	4.7	5.7
GP FL 5	3.6	3.5
GP FL 6	4.9	5.9
I QK FL	4.0	5.0
MO (A)	5.0	5.9
E INT 6	5.3	6.4
OCC 4	5.5	6.5
Wt. Average	3.2	3.1

6.4 Buoy Roll Amplitude and Frequency Effects

Strong dependence of POD and POR on roll amplitude is immediately apparent from the data presented thus far. As Table 5 shows, significant nominal range degradation results with less than 10 degrees rms roll using standard aids-to-navigation hardware. In general, as rms roll amplitude increases, POD and POR decrease, but nonlinearly. Although insufficient data exist to determine this exact relationship, Table 5 indicates nominal range degradation may approach a limiting value beyond about 8 degrees rms roll.

Buoy roll power spectral densities were determined directly from angle versus time recordings. Roll PSDs are given in Figures 18 through 22. Prior to applying the Fourier transform, time domain data were adjusted in three ways: a) any DC offset was removed or minimized, b) high irequency noise was removed by convolution, and c) leakage of high frequencies into the low frequency range

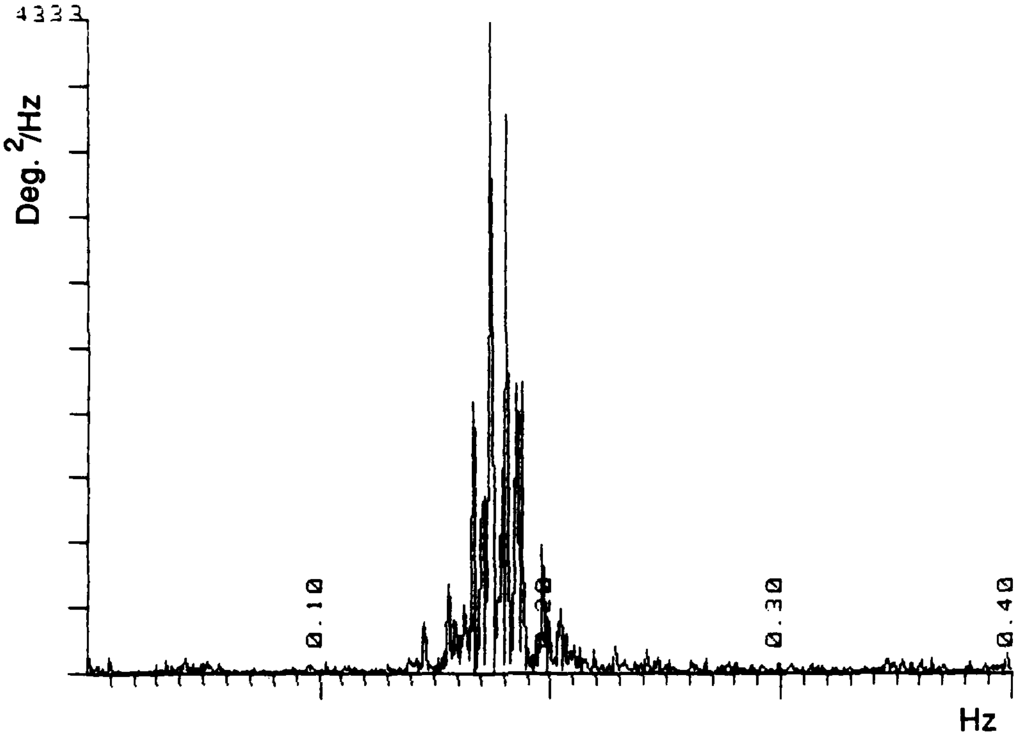


FIGURE 18. 8 x 26 Buoy Roll PSD (9.9 Deg. rms)

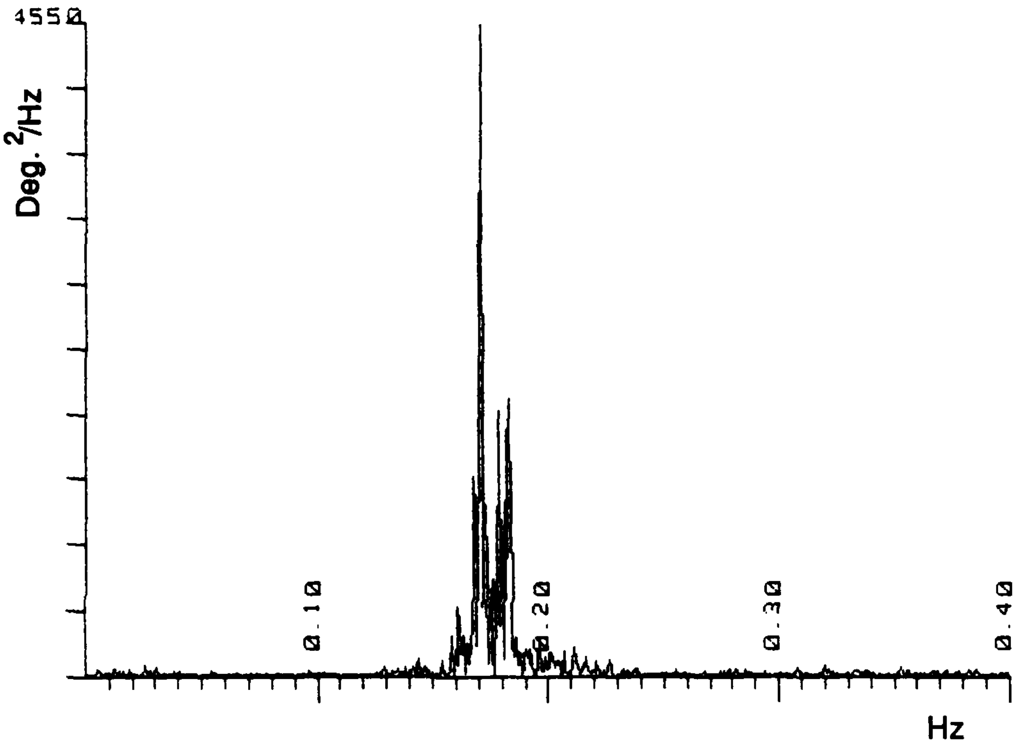


FIGURE 19. 8 x 26 Buoy Roll PSD (8 Deg. rms)

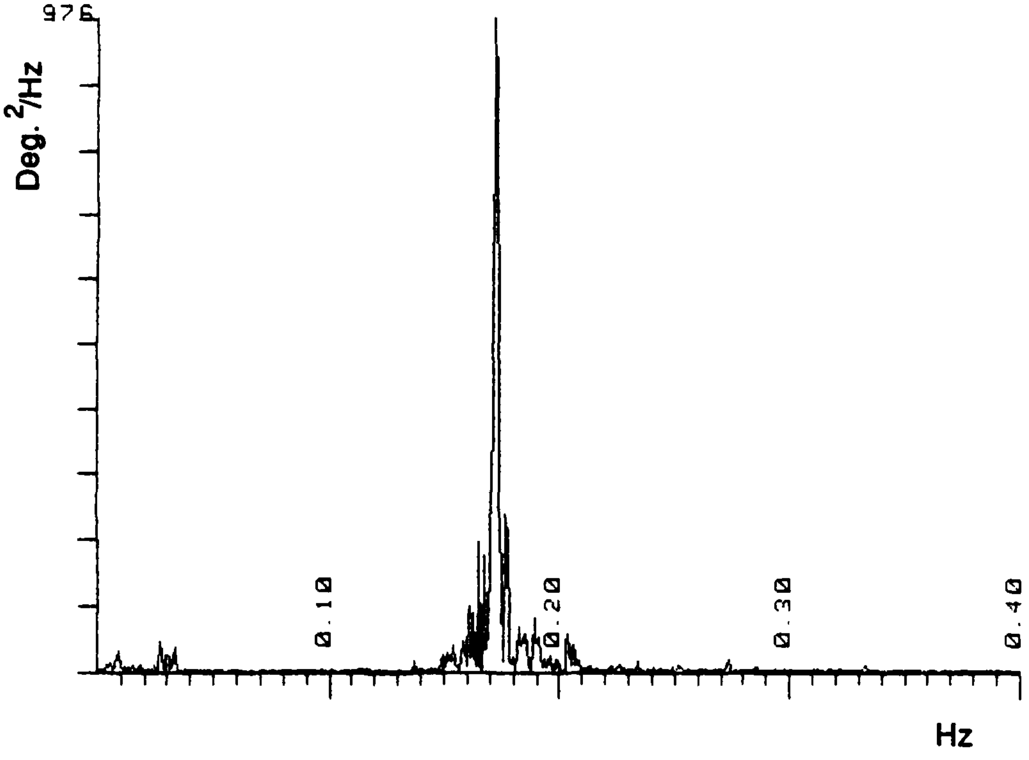


FIGURE 20. 8 x 26 Buoy Roll PSD (3.3 Deg. rms)

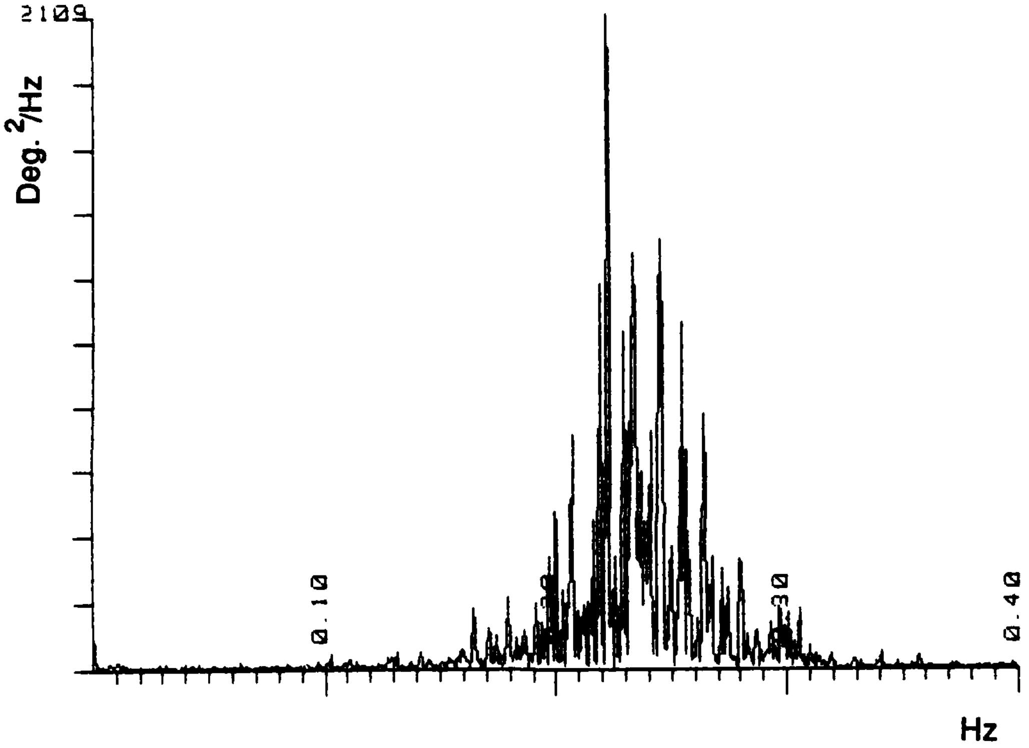


FIGURE 21. 5 x 11 Buoy Roll PSD (8.4 Deg. rms)

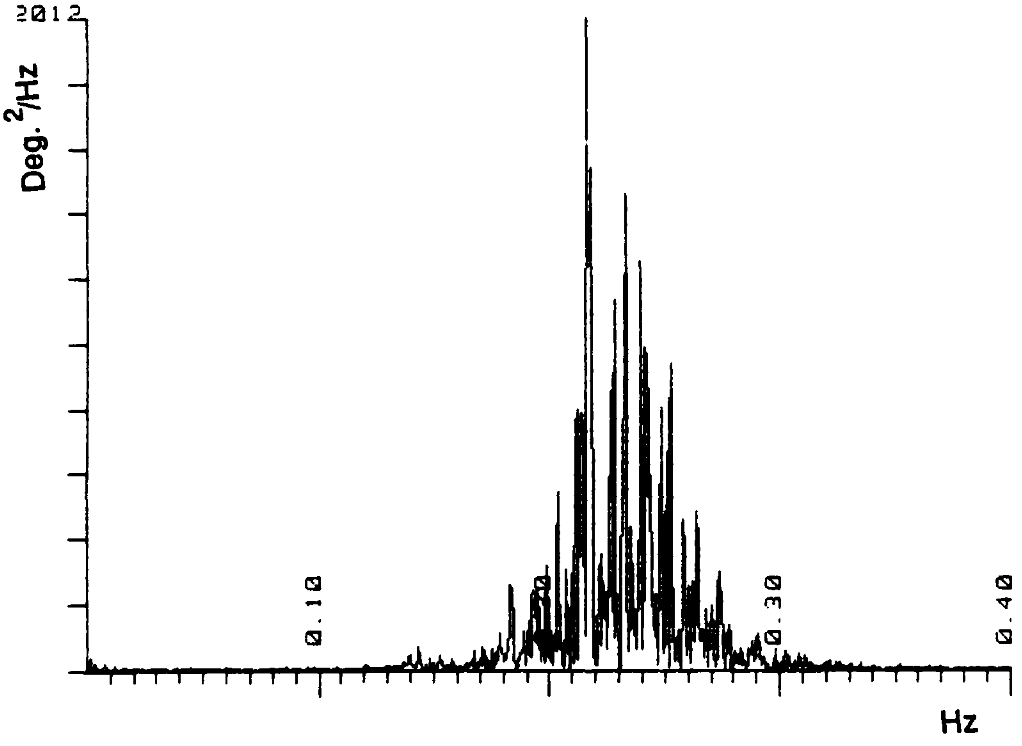


FIGURE 22. 5 x 11 Buoy Roll PSD (8.2 Deg. rms)

was minimized by multiplying with either Hamming, Hanning, or Hanning squared windows (Ramirez (1985)).

Figures 18 through 22 show power spectral densities (PSDs) of the five buoy roll recordings. Ordinate values on PSD plots are in units of degrees squared per Hertz[3]. Rms roll amplitude is proportional to the square root of the area under the PSD curve (Appendix A). Thus roll amplitudes are reduced if bandwidth and peak response of the buoy transfer function are reduced. This is analogous to electrical bandpass filters. Reducing bandwidth reduces the amount of noise (wave) power transferred to system output.

PSDs show strong components near natural roll frequencies. Figure 18 shows a peak roll component at 0.18 Hz giving a roll period of 5.6 seconds. This conforms closely to Milne, Hsu, and Smith (1975) who show roll periods for the 8x26 buoy of about 5.5 seconds.

Roll amplitudes are reduced if buoys are designed with natural roll frequencies different from frequencies of sea spectra peaks. A cursory review of the literature shows that both the 5x11 and 8x26 buoys have natural roll periods near sea spectra peaks. Colburn (1974) showed PSD sea spectra peaks at wave periods from 5.5 to 15.5 seconds off the California coast. Yamazaki and Herbich (1985) show spectral density peaks at wave periods from 3.5 to 7.6 seconds in the Gulf of Mexico near Galveston, Tx. Schwab, Bennett, and Liu (1984) observed wave periods between 2 to 6 seconds in Lake Erie.

6.5 Roll Period / Flash Period Correlation Effects

Roll amplitude effects were expected and are easily understood. Less apparent is the dependence of POD and POR on actual frequency components present in buoy motion. Compare the results of the 8x26 with 8 degrees roll and two 5x11 buoy recordings in Tables 3 and 4. The two 5x11 buoy recordings both have rms roll amplitudes greater than 8 degrees, yet they provide greater average detection and recognition distances than the 8x26 buoy with 8-degree rms roll. This indicates POD and POR depend on buoy roll spectra frequencies.

Figures 18 through 22 show differences in frequency composition between the two buoys. Bandwidths indicate buoy roll is

[3] Actual frequency resolution equals the sampling rate divided by the number of points transformed. Sampling rate and number of points were 5 Hz and 8192 points for three 8x26 buoy recordings and one 5x11 buoy, giving a frequency resolution of 0.00061 Hz. Sampling rate and number of points for the remaining 5x11 buoy recording were 2 Hz and 4096, giving a frequency resolution of 0.00049 Hz.

correlated over a time interval of about 16 seconds for the 8x26 buoy and about 10 seconds for the 5x11 buoy. Buoy motion is not completely random and correlation with the flash characteristic will affect detection and recognition probabilities.

To illustrate this consider the ideal case of a buoy rolling with a single frequency f_1 and constant amplitude. Two possible outcomes are: (1) the light flashes when buoy is vertical and is detected, and (2) the light flashes when buoy is at maximum roll angle and is not detected. A continuum of outcomes lie between these two, where detection may or may not occur, depending on lens vertical divergence and distance. If the light flashes with frequency f_2 then outcome 1 (simultaneous flash and vertical buoy) occurs every

$$T = \frac{1}{| f_1 - f_2 |} \qquad (4)$$

seconds. In this special case, if observation time is greater than T as given in equation (4), detection probability is 100%. Detection probability decreases as T becomes greater than observation time. The roll period for the 8x26 with 10 degrees rms roll is about 5.7 seconds. Thus we expect a decrease in POD as flash periods approach 5.7 seconds.

To test this, 90% POD distances (for 12-second observation time) were determined for five additional nonstandard flash characteristics on the 8x26 buoy with 10-degree rms roll. The characteristics had flash periods from 4.5 seconds to 7 seconds, but all had flash pulse durations of 0.6 seconds to make a direct comparison with the FL 6 characteristic. Results, tabulated in Table 8, show a minimum 90% POD distance when the flash characteristic period matches the buoy roll period.

Similar results occur with the 5x11 buoy which has a roll period of about 4.6 seconds. It produced higher PODs for the FL6 characteristic than for the FL4 for observation times greater than 12 seconds.

Equation (4) holds as long as buoy roll is correlated to a constant frequency. As buoy roll becomes more random (autocorrelation time decreases) detection probability increases. This accounts for the slightly higher average POD distance for the 5x11 buoy than on the 8x26 buoy with 8 degrees rms roll (Table 3).

On the other hand, correlation between buoy roll and flash characteristic improves recognition probability. Recognition requires (according to definition) detection of at least two consecutive flashes. A rolling 8x26 buoy crosses vertical approximately every 2.9 seconds. A QK FL characteristic flashing every second is less likely to have two consecutive flashes occur at vertical than a FL 2.5 or FL 5.5. We should expect probabili-

TABLE 8

90% POD and POR Distances
Versus Flash Period Correlation

Flash Charac.	12-sec/90% POD Distance	14-sec/90% POR Distance
FL 7 (0.6)	1.8 n.miles	1.06 n.miles
FL 6.5 (0.6)	1.9	1.11
FL 6 (0.6)	1.8	1.11
FL 5.5 (0.6)	1.5 *	1.18 *
FL 5 (0.6)	1.9	1.14
FL 4.5 (0.6)	2.6	1.33
FL 2.5 (0.3)	2.2	2.00 *
QK FL	5.1	1.42

* Maximum or minimum

ty of recognition to peak as flash period approaches an integer multiple of one-half the buoy roll period. This is confirmed by the 90% POR distances (for 14-second observation time) listed in Table 8.

7. DISCUSSION

7.1 Validity of Assumptions

Analysis of POR assumes recognition occurs with two complete and consecutive flash periods. Recognition is more complex than this definition of POR implies. Color, familiarity with the waterway, and intelligent elimination of possibilities also affect recognizing a flash characteristic. This definition is applied uniformly to all characteristics, although it allows confusion between the QK FL and I QK FL characteristics. Requiring seven consecutive periods of QK FL would distinguish it from the I QK FL, and would also significantly lower its POR values. On the other hand, if I QK FL and QK FL characteristics were never placed together in a channel, the mariner would not have to distinguish between the two.

The definition of POR was chosen to model a task more difficult than detection, but one which was simply calculated by a computer in a reasonable amount of time. POR values may not accurately model recognition, since it is such a complex process, but they do place a lower limit on detection.

Analyses assumed the mariner knows where to look for the buoy. Thus every flash pulse which produces at least threshold illuminance at the observer's position is assumed detected. If the mariner does not know where to look for the buoy, PODs and PORs would be lower.

Ignoring lamp nigrescence time in the calculations was also driven by computation time and algorithm complexity. Figure 7 indicates that highest rotational velocities may produce intensity rise and fall times on the order of lamp nigrescence time. Much of the intensity modulation is slower than this however. Including nigrescence time in the calculations would result in slightly lower probabilities. Larger reductions would occur with shorter pulse duration characteristics such as the QK FL and FL 2.5.

The possibility of false flashing within a single pulse due to buoy rolling was ignored. Pulse duration was used to define the limits of integration. This assumption results in slightly higher EFI values than those using the Blondel-Rey-Douglas method. However, both methods are still only approximations to a psychophysical process. Using the Blondel-Rey-Douglas method would significantly increase an already very long computational process.

No assumption is made or implied that the data collected represent average conditions on U.S. waterways. The sampling is far too limited to represent even local conditions. However, Milne, Hsu, and Smith (1975) show that 10-degree rolls on 8x26 buoys are not uncommon. They recorded up to 32-degree rolls on an 8x26 in a sea state with wave heights of 1.2 feet rms.

7.2 Scope of Research

The purpose of this study was to determine if a POD problem existed due to buoy motion and to outline a method to quantify the problem. Although a greater number of buoy roll recordings were expected, the goals of the study have been accomplished without them.

This study does not determine (nor was it intended to determine) which is the best flash characteristic, best lens divergence, and best buoy on U.S. waterways. These questions would be answered by a parametric study, which this was not. This study only provides the tools and outlines the methodology to perform a parametric study of lighted buoy systems.

8. CONCLUSIONS

Nominal range is not a realistic measure of detection range of a lighted buoy. Flash characteristic, sea state, lantern divergence, and observation time influence detection and recognition range of a floating aid to navigation. For the relatively sheltered conditions studied here, some startling facts were discovered. Among them are:

1. Detection ranges (90% Probability of Detection) are as little as 30% of the published Nominal Range.

2. The most commonly used characteristic, FL 4(.4), on an 8x26 buoy has an effective range of only 1/3 of the published nominal range.

3. Doubling lantern vertical divergence will nearly double effective range of floating aids to navigation with 10% duty cycles.

4. Not only does amplitude of buoy roll degrade signal effectiveness, but so does correlation between period of buoy roll and flash rhythm.

The following general observations and trends are derived from the results:

1. Significant nominal range degradation results with less than 10 degrees rms buoy roll (section 6.4)

2. Increasing lens vertical divergence can significantly increase POD of buoy lights (section 6.3)

3. Changing a flash characteristic can significantly increase POD of a buoy light (sections 6.2, 6.5)

4. POD and POR decrease with increasing observation distance (section 6.1)

5. POD and POR increase with increasing observation time (section 6.1)

A unique method of determining probabilities of detecting and recognizing flashing lights on rolling buoys has been demonstrated. The method takes into account observer distance, observation time, lens vertical divergence, flash characteristic, and buoy roll spectra. All five variables affect POD and POR.

9.0 RECOMMENDATIONS

A better measure of buoy light performance than nominal range should be investigated. The 90% POD distance (for 12-second observation time) is proposed and used in this study. Before this measure can be adopted, further research on probabilities of detecting and recognizing flashing lights on rolling buoys is needed. Recommendations for further research are as follows.

> Investigate 90% POD distance for use as a measure of buoy light performance. Ninety percent (90%) POD allows engineers to optimize system parameters. The 90% POD distance (or any other criterion POD distance) could be used without specific knowledge of local wave conditions by adopting standard or average sea spectra for the United States or several geographical regions.
>
> Collect additional data to extend the results of this study to cover a more diverse range of sea conditions. The assumption that a single axis of measurement is adequate should be tested by measuring roll angles in two mutually perpendicular directions. Video recording may prove the best method for obtaining these new data.
>
> Examine the effect of increased vertical divergence on POD. Explore methods of accomplishing this without designing a new lens (i.e. frosted lamps, longer lamp filaments, new lamps).
>
> Formulate a method to predict POD and POR as a function of five variables: observation time, observation distance, lens divergence, flash characteristic, and roll spectra.
>
> Investigate resistance of various buoy designs to motion in a variety of sea states.

10. REFERENCES

Bhattacharyya, R., Dynamics of Marine Vehicles, John Wiley & Sons, 1978

Colburn, W. E., Acquisition and Analysis of Ocean Wave Data Recorded on a Spar-Buoy Mounted Digital Cassette Recorder, Master of Science Thesis, University of Rhode Island, 1974

Feingold, H., Ritter, D., Tozzi, J., "Probabilities of Detection and Identification of Navigation Buoy Light Signals", David Taylor Naval Ship Research and Development Center, Bethesda, Md., Report No. 77-0031, February 1977

Hirsch, R.A., A Comparative Analysis of the Probabilities of Detection of Buoy Signal Lights, Report No. EW-8-78, Mechanical Engineering Dept., US Naval Academy, May 1978

IALA, Recommendations on the Determination of the Luminous Intensity of a Marine Aid-to-Navigation Light, IALA Bulletin No. 75-1978-3, International Association of Lighthouse Authorities, Paris, France, December 1977

Milne, D.T., Hsu, S.K., Smith, W.E., "U.S. Coast Guard Moored Buoy Data Handling and Analysis," David W. Taylor Naval Ship Research and Development Center, Report No. SPD-657-01, December 1975

Price, D., "Buoy Response Amplitude Operators Obtained From Step Response Tests," Offshore Technology Conference, OTC 2467, pp. 459-478, May 1976

Ramirez, R., The FFT Fundamentals and Concepts, Prentice-Hall, 1985

Schwab, D.J., Bennett, J.R., Liu, P.C., "Application of a Simple Numerical Wave Prediction Model," Journal of Geophysical Research, Vol 89, No. C3, pp 3586-3592, May 20, 1984

U.S. Coast Guard (G-EOE), Aids To Navigation Technical, COMDTINST M16500.3, 1983

U.S. Coast Guard (G-EOE), Luminous Intensities of Aids To Navigation Lights, COMDTINST M16510.2, 1984

Yamazaki, H., Herbich, J.B., Nonparametric and Parametric Estimation of Wave Statistics and Spectra, TAMU-SG-86-202, COE Report No. 279, Texas A&M University, October 1985

APPENDIX A

SPECTRAL ANALYSIS OF BUOY ROLL

I. LINEARITY AND TRANSFER FUNCTION

If a buoy is assumed to be a linear system then in the time domain the buoy's output response y(t) equals the convolution of the buoy's impulse response h(t) and a wave input function x(t). Convolution in the time domain is equivalent to multiplication in the frequency domain. Thus a buoy's roll spectra Y(f) equals the product of the complex transfer function H(f) and the sea spectra X(f). The square of the absolute value of H(f) is frequently called the response amplitude operator (RAO), (although some also define |H(f)| as the RAO e.g. Price, Milne, et. al.). The power spectral density (PSD) of buoy roll is the square of the absolute value of the roll spectra. Defining F{ h(t) } to mean the Fourier transform of h(t) these relationships are given by:

$$H(f) = F\{ h(t) \} \qquad \text{(A1)}$$

$$Y(f) = F\{ y(t) \} \qquad \text{(A2)}$$

$$X(f) = F\{ x(t) \} \qquad \text{(A3)}$$

$$Y(f) = X(f)\, H(f) \qquad \text{(A4)}$$

$$PSD = |Y(f)|^2 \qquad \text{(A5)}$$

Linearity requires a buoy to satisfy two criteria: 1) any frequency component of buoy roll must be a linear function of wave amplitude of the same frequency, and 2) the buoy's response to any individual wave frequency component is independent of it's response to any other wave frequency component. Price (1976) demonstrates this is practically true for at least some Coast Guard buoys. Bhattacharyya (1978) states that linearity fcr floating vessel motion is a valid assumption if the amplitudes are moderate. Although responses are actually nonlinear, nonlinearities can be ignored in practice.

It is difficult to provide an impulse input to a buoy and measure its impulse response directly. However a step input can be simulated by pulling the buoy over and quickly releasing it. The resulting time series response may be normalized by the initial starting angle to simulate a unit step response. This response is actually a negative step response as the input force (or buoy angle) is unity prior to time zero and zero afterwards. The relationship between the impulse response, h(t), and a negative step response may be determined by starting with the convolution integral.

$$y(T) = \int_{-\infty}^{\infty} x(T-t)\, h(t)\, dt \qquad \text{(A6)}$$

If the input is a negative unit step function:

$$x(t) = \begin{cases} 1 \text{ if } t < 0 \\ 0 \text{ if } t > 0 \end{cases} \qquad (A7)$$

then the output y(t) is the negative unit step response. In the convolution integral above, the input function is reversed. Thus x(T-t) is a positive unit step beginning at T:

$$x(T-t) = \begin{cases} 1 \text{ if } t > T \\ 0 \text{ if } t < T \end{cases} \qquad (A8)$$

Equation A8 implies that integral A6 is nonzero only for time greater than T. Thus the convolution integral reduces to

$$y(T) = \int_{T}^{\infty} h(t)\, dt = H(\infty) - H(T) = 0 - H(T) \qquad (A9)$$

Taking the derivative of equation A9 with respect to time shows the negative unit step response is related to the impulse response by:

$$\frac{d\ y(t)}{dt} = -\ h(t) \qquad (A10)$$

The derivative theorem for Fourier transforms states that the transform of the derivative of y(t) equals the product of Y(f) and the imaginary radian frequency. Multiplication by frequency enhances higher frequencies and attenuates lower frequencies. Multiplication by the square root of negative one implies a 90-degree phase shift. Thus the transfer function H(f) may be determined from the Fourier transform of the negative step response Y(f) according to:

$$H(f) = -\ j\ 2\pi\ f\ Y(f) \qquad (A11)$$

II. POWER SPECTRAL DENSITY OF BUOY ROLL

According to equations A1 through A5, power spectral density (PSD) curves can be predicted from transfer functions and sea spectra. The usefulness of PSD can be seen from Parseval's theorem which states that the energy in a waveform x(t) in the time domain equals the energy of its transform X(f) in the frequency domain. For discrete or sampled functions Parseval's theorem is given by:

$$\sum_{k=0}^{N-1} [\ x(k)\ ]^2 = \sum_{n=0}^{N-1} [\ X(n)\ ]^2\ /\ N \qquad (A12)$$

if the discrete Fourier transform is defined as:

$$X(n) \quad = \quad \sum_{k=0}^{N-1} x(k) \ \exp(-j2\pi nk/N) \qquad \text{(A13)}$$

and its inverse as:

$$x(k) \quad = \quad N^{-1} \sum_{n=0}^{N-1} X(n) \ \exp(\ j2\pi nk/N) \qquad \text{(A14)}$$

The integral of the PSD function from negative cutoff frequency to positive cutoff frequency equals the variance (mean square) of the roll amplitude. (Cutoff frequency is one half sampling rate. Only positive frequencies were plotted since the negative half is a mirror image of the positive half of the PSD.) Thus PSDs indicate the energy content of various frequency components of roll. The range or bandwidth of predominant frequencies gives an estimate of output frequency correlation time.

Research showed that roll PSD is one of the variables affecting detection and recognition probabilities, due to correlation effects with the flash characteristic. PSD can be reduced to a single number via the cross correlation coefficient. Additional research is needed to determine the relationship between the correlation coefficient and POD / POR. Since wave PSDs have been measured for numerous oceans and waterways throughout the world, spectral analysis may prove to be a useful tool for determining optimal flash characteristics and optimal buoy designs.

[BLANK]

APPENDIX B

Programs for Data Collection and Analysis

Program RECORD_BUOY_MOTION

```
PROGRAM RECORD_BUOY_MOTION (INPUT, OUTPUT);
{Records buoy roll angle as a function time and stores data on silo mass
storage device. Plots angle in real time on CRT. by Dan Brown, APR 87}
IMPORT
  DGL_LIB,                         {get graphics routines}
  HPIB_0,
  GENERAL_0,
  GENERAL_1,
  GENERAL_2,
  SYSGLOBALS,
  SYSDEVS,
  IOCOMASM,
  IODECLARATIONS;
CONST
  HPIB          = 7;            { hpib interface select code }
  GPIO          = 12;           { gpio interface select code }
  HP5334A       = 704;          { hp univ. counter isc }
  K485          = 722;          { auto. picometer ics }
  MIDNIGHT      = 8640000;      { centi seconds per day }
  ESC           = CHR(27);       {constants for silo commands}
  CrtAddr       = 3;            { address of internal crt }
  ControlWord   = 0;            { device control; 0 for crt }
  OPEN_LENS     = 3;            { i/o control: open lens, motor on }
  CLOSE_LENS    = 4;            { i/o control: close lens, motor off }
  MAXANGLE      = 45;           {max degrees shown on CRT plot }
  WAVECAL       = 3379600;      {wave rider calibration, ft/amp }
TYPE
  MSG_TYPE      = STRING[255];
  DIR_TYPE      = ( RIGHT, LEFT );        {enumerated direction type}
VAR
  STARTTIME     : INTEGER;      { daily start time for data collection}
  STOPTIME      : INTEGER;      { daily stop time for data collection}
  NEXTONTIME    : INTEGER;      { next start time for data collection }
  ONDURATION    : INTEGER;      { time stepper motor is on }
  OFFDURATION   : INTEGER;      { rest time between blocks of data }
  FINALSTOPDATE,
  NEXTSTARTDATE,
  NEXTSTOPDATE,
  TODAYSDATE    : DATEREC;
  SAMPLERATE    : REAL;      { rate buoy angle is measured }
  SIGNALFREQ    : REAL;         { signal frequency from PMT detector }
  OFFSET        : REAL;         { uc_5334a counter time when buoy vertical }
  I             : INTEGER;
  ANS           : CHAR;
  TIME          : TIMEREC;
  PTR           : TEXT;
  F             : TEXT;
  BUOYTYPE      : STRING [15];

(*****************************************************)
(*                                                  *)
(*                PROCEDURE GET_DATE                *)
(*                                                  *)
(* Displays prompt, reads date input and checks     *)
(* format of input.  Returns date to SET_DATES.     *)
(*                                                     *)
(*******************************************************)
PROCEDURE GET_DATE (VAR DATE : DATEREC; MSG : MSG_TYPE);
VAR
  J               : INTEGER;      { loop counter }
  DATEVALID       : BOOLEAN;      { date status  }
  DSTR            : STRING[40];
  DAY             : 1..31;
  MONTH           : 1..12;
  YEAR            : 86..88;   {PROGRAM WILL BOMB IN 89!}
  DELIMIT         : CHAR;
BEGIN
REPEAT
    PAGE;
    DATEVALID := TRUE;
    FOR J := 1 TO 10 DO WRITELN;
    WRITELN (MSG);
    WRITELN;
    WRITE('The current date is:         ');
    WRITELN(DATE.MONTH:2,'/',DATE.DAY:2,'/',DATE.YEAR:2);
    WRITELN;
    WRITELN('Enter new date in format:  mm/dd/yy');
    READLN( DSTR );
    IF STRLEN( DSTR ) > 0 THEN
        BEGIN
        TRY
            STRREAD(DSTR,1,I,MONTH,DELIMIT,DAY,DELIMIT,YEAR);
            DATE.MONTH := MONTH;
            DATE.DAY := DAY;
            DATE.YEAR := YEAR;
        RECOVER
            BEGIN
            PAGE;
            WRITELN('Unrecognized date format.  Try again.');
            WRITELN('Example: 07/30/86');
            DATEVALID := FALSE;
            END;
        END;
UNTIL DATEVALID;
END;  (* PROCEDURE GET_DATE *)

(*******************************************************)
(*                                                     *)
(*                PROCEDURE SET_DATES                  *)
(*                                                     *)
(*  Sets dates (todaysdate, startdate, stopdate).      *)
(*  Promts for date input.  Displays current value.    *)
(*  If OK then <Return> accepts current value for the  *)
(*  date.  Each date is obtained via procedure         *)
(*  GET_DATE.                                          *)
(*                                                     *)
(*  units called        : SYSDATE ( in SYSDEVS )       *)
(*                        SETSYSDATE ( in SYSDEVS )    *)
(*                        Procedure GET_DATE           *)
(*                                                     *)
(*******************************************************)
```

```
PROCEDURE SET_DATES( VAR TODAYSDATE, STARTDATE, STOPDATE : DATEREC);
VAR
  MESSAGE : STRING[80];
BEGIN
MESSAGE := 'ENTER TODAYS DATE';
GET_DATE( TODAYSDATE, MESSAGE );
SETSYSDATE (TODAYSDATE);          { set system date }
MESSAGE := 'ENTER STARTING DATE';
GET_DATE( STARTDATE, MESSAGE );
MESSAGE := 'ENTER PROGRAM STOP DATE';
GET_DATE( STOPDATE, MESSAGE );
END;   (* PROCEDURE SET_DATE *)

(******************************************************)
(*                                                    *)
(*                PROCEDURE CONVERT_TIMES             *)
(*                                                    *)
(*  Converts time in centiseconds to a time record if *)
(*  direction is "right".  If direction is left then  *)
(*  converts timerec to centiseconds.                 *)
(*                                                    *)
(******************************************************)
PROCEDURE CONVERT_TIMES( VAR TIMECSEC : INTEGER;
                        DIRECTION : DIR_TYPE;
                        VAR TIMERECORD : TIMEREC );
VAR  CSEC : INTEGER;
BEGIN
CASE DIRECTION OF
  RIGHT : BEGIN         { convert centiseconds to timerec }
          CSEC := TIMECSEC;
          TIMERECORD.HOUR := CSEC DIV 360000;
          CSEC := CSEC MOD 360000;
          TIMERECORD.MINUTE := CSEC DIV 6000;
          TIMERECORD.CENTISECOND := CSEC MOD 6000;
          END;
   LEFT : BEGIN         {convert timerec to centiseconds }
          TIMECSEC := 360000*TIMERECORD.HOUR + 6000*TIMERECORD.MINUTE +
                           TIMERECORD.CENTISECOND;
          END;
   END; {CASE}
END;    { CONVERT_TIMES }

(******************************************************)
(*                                                    *)
(*                PROCEDURE GET_TIME                  *)
(*                                                    *)
(*  Displays prompt and takes input for time.         *)
(*  If <CR> then accepts displayed time for input.    *)
(*                                                    *)
(******************************************************)
PROCEDURE GET_TIME (VAR CENTISECONDS : INTEGER;
                    MSG : MSG_TYPE);
VAR
  J              : INTEGER;      { loop counter }
  TIMEVALID      : BOOLEAN;
  TSTR           : STRING[8];  { input buffer }
  DELIMIT       : CHAR;         { separator     }
  TIME          : TIMEREC;
BEGIN
REPEAT
    TIMEVALID := TRUE;
    PAGE;
    FOR J := 1 TO 10 DO WRITELN;  { set up prompt display and read screen }
    WRITELN (MSG);        { write what time is for }
    WRITELN;
    WRITE ('It is now set to:           ');
    CONVERT_TIMES( CENTISECONDS,RIGHT,TIME );
    WRITELN (TIME.HOUR:2,':',TIME.MINUTE:2,':',TIME.CENTISECOND DIV 100:2);
    WRITELN;
    WRITE ('Enter the new time in the form: hh:mm:ss ');
    READLN (TSTR);
    IF STRLEN (TSTR) > 0 THEN
        BEGIN
        TRY
          STRREAD(TSTR,1,J,TIME.HOUR,DELIMIT,TIME.MINUTE,DELIMIT,
                             TIME.CENTISECOND);
          TIME.CENTISECOND := TIME.CENTISECOND * 100;
          CONVERT_TIMES( CENTISECONDS,LEFT,TIME );
        RECOVER
          BEGIN
          PAGE;
          WRITELN ('Urecognized time format.  Try again.');
          WRITELN('For example: 12:34:56');
          TIMEVALID := FALSE;
          END;
        END;
UNTIL TIMEVALID;
END;

(******************************************************)
(*                                                    *)
(*                  PROCEDURE SET_TIMES               *)
(*                                                    *)
(*  Sets the various times used by program.  Prompts  *)
(*  for input by user.  Times are all in centiseconds *)
(*                                                    *)
(*  units called        : SETSYSTIME ( in SYSDEVS )   *)
(*                        Procedure GET_TIME          *)
(*                                                    *)
(******************************************************)
PROCEDURE SET_TIMES( VAR STARTTIME,
                         STOPTIME,
                         ONDURATION,
                         OFFDURATION : INTEGER );
VAR
  I             : INTEGER;      { LOOP COUNTER }
  MESSAGE       : MSG_TYPE;
  CURRENTTIME   : INTEGER;
  TIME          : TIMEREC;
BEGIN
MESSAGE := 'ENTER CURRENT TIME OF DAY';
CURRENTTIME := SYSCLOCK;
```

```
GET_TIME ( CURRENTTIME, MESSAGE);
CONVERT_TIMES( CURRENTTIME,RIGHT,TIME );
SETSYSTIME(TIME);
MESSAGE := 'ENTER DAILY START TIME FOR DATA COLLECTION PHASE';
GET_TIME (STARTTIME, MESSAGE);
MESSAGE := 'ENTER DAILY STOP TIME FOR DATA COLLECTION PHASE';
GET_TIME (STOPTIME, MESSAGE);
MESSAGE := 'ENTER TIME LENGTH OF A DATA BLOCK';
GET_TIME (ONDURATION, MESSAGE);
MESSAGE := 'ENTER TIME LENGTH BETWEEN BLOCKS OF DATA (REST TIME)';
GET_TIME (OFFDURATION, MESSAGE);
END;    (* PROCEDURE SET_TIMES *)

(******************************************************)
(*                                                    *)
(*                PROCEDURE GET_NUMBER                *)
(*                                                    *)
(*  Sets up display and prompts operator for input    *)
(*  of a real number.  Passes input back to           *)
(*  SET_DATARATES.                                    *)
(*                                                    *)
(******************************************************)
PROCEDURE GET_NUMBER (VAR NUM : REAL; MESSAGE : MSG_TYPE);

VAR
  NSTR          : STRING[8];  { input buffer }
  I             : INTEGER;      { counter      }
  VALIDNUMBER   : BOOLEAN;
BEGIN
REPEAT
    PAGE;
    VALIDNUMBER := TRUE;
    FOR I:=1 TO 10 DO WRITELN;
    WRITELN( MESSAGE );
    WRITELN('It is currently set to ',NUM:6:4);
    READLN( NSTR );
    IF STRLEN (NSTR) > 0 THEN
        BEGIN
        TRY
            STRREAD (NSTR,1,I,NUM);
        RECOVER
            BEGIN
            PAGE;
            WRITELN ('ERROR : Invalid data entry. Try again.');
            VALIDNUMBER := FALSE;
            END;
        END;
UNTIL VALIDNUMBER;
END;  (* PROCEDURE GET_NUMBER *)

(******************************************************)
(*                                                    *)
(*              PROCEDURE SET_DATARATES               *)
(*                                                    *)
(*  Prompts operator for buoy angle sampling rate,    *)
(*  counter time offset, and PMT signal frequency.    *)
(*  Prompts are repeated until valid response.       *)
(*   <CR> will take the current value being displayed *)
(*                                                   *)
(*  units called : GET_NUMBER                        *)
(*                                                   *)
(*  limits       : off set constant >= 0             *)
(*                 frequency constant > 0            *)
(*                                                   *)
(*****************************************************)
PROCEDURE SET_DATARATES (VAR SAMPLERATE : REAL;
                         VAR OFFSET, SIGNALFREQ : REAL );
VAR
    I             : INTEGER;      { loop counter }
    NUM           : REAL;
    MESSAGE       : MSG_TYPE;
    CRLF          : STRING[2];
BEGIN
CRLF := CHR(13)+CHR(10);       {carriage return line feed}
MESSAGE := 'INPUT THE BUOY ANGLE SAMPLING RATE IN HERZ'+CRLF+
   '3 Hz or less -> both buoy and waverider signals stored'+CRLF+
   'greater than 3 Hz to 8 Hz -> only buoy signal is stored'+CRLF+
   '..... default is 8 Hz';
GET_NUMBER(SAMPLERATE,MESSAGE);
MESSAGE := 'INPUT COUNTER TIME (IN SECONDS) WHEN BUOY IS VERTICAL';
GET_NUMBER( OFFSET, MESSAGE );
MESSAGE := 'INPUT PMT SIGNAL FREQUENCY (IN HERTZ)';
GET_NUMBER( SIGNALFREQ, MESSAGE );
END;

(*****************************************************)
(*                                                   *)
(*              PROCEDURE SHOW_SETTINGS              *)
(*                                                   *)
(*  Displays all input dates, times, and constants,  *)
(*  and prompts user to accept or reject values.     *)
(*  Must respond with a Y or a N.  This response is  *)
(*  returned to the calling unit.                    *)
(*                                                   *)
(*****************************************************)
PROCEDURE SHOW_SETTINGS (VAR ANS : CHAR;
                         TODAYSDATE,NEXTSTARTDATE,FINALSTOPDATE : DATEREC;
                         STARTTIME,STOPTIME,
                         ONDURATION, OFFDURATION : INTEGER;
                         SAMPLERATE, OFFSET, SIGNALFREQ : REAL);
VAR
  I             : INTEGER;      { loop counter }
  CSEC          : INTEGER;
  TIME          : TIMEREC;
BEGIN
PAGE;
FOR I := 1 TO 7 DO WRITELN;
WRITE('TODAYS DATE IS ................................. ');
WRITELN(TODAYSDATE.MONTH:1,'/',TODAYSDATE.DAY:1,'/',TODAYSDATE.YEAR:1);
WRITE('CURRENT TIME OF DAY IS ......................... ');
SYSTIME( TIME );
WRITELN(TIME.HOUR:2,':',TIME.MINUTE:2,':',TIME.CENTISECOND DIV 100:2);
```

```
WRITE('DATA COLLECTION START DATE IS .................... ');
WRITELN(NEXTSTARTDATE.MONTH:2,'/',NEXTSTARTDATE.DAY:2,'/',
            NEXTSTARTDATE.YEAR:2);
WRITE('DAILY STARTTING TIME IS .......................... ');
CONVERT_TIMES( STARTTIME,RIGHT,TIME );
WRITELN(TIME.HOUR:2,':',TIME.MINUTE:2,':',TIME.CENTISECOND DIV 100:2);
WRITE('DATA COLLECTION STOP DATE IS ..................... ');
WRITELN(FINALSTOPDATE.MONTH:2,'/',FINALSTOPDATE.DAY:2,'/',
            FINALSTOPDATE.YEAR:2);
WRITE('DAILY STOP TIME IS ............................... ');
CONVERT_TIMES( STOPTIME,RIGHT,TIME );
WRITELN(TIME.HOUR:2,':',TIME.MINUTE:2,':',TIME.CENTISECOND DIV 100:2);
WRITE('LENGTH OF TIME FOR CONTINUOUS BLOCK OF DATA IS ... ');
CONVERT_TIMES( ONDURATION,RIGHT,TIME );
WRITELN(TIME.HOUR:2,':',TIME.MINUTE:2,':',TIME.CENTISECOND DIV 100:2);
WRITE('OFF TIME BETWEEN DATA BLOCKS IS .................. ');
CONVERT_TIMES( OFFDURATION,RIGHT,TIME );
WRITELN(TIME.HOUR:2,':',TIME.MINUTE:2,':',TIME.CENTISECOND DIV 100:2);
WRITE('DATA SAMPLING RATE IS ............................ ');
WRITELN( SAMPLERATE:2:2 );
WRITE('COUNTER TIME WITH BUOY VERTICAL IS ............... ');
WRITELN(OFFSET:4:3);
WRITE('PMT SIGNAL FREQUENCY IS .......................... ');
WRITELN(SIGNALFREQ:4:3);
WRITELN;
WRITELN;
WRITE ('Do you want to change any values?   Y/N    ');
REPEAT
    READ (ANS);
UNTIL (ANS = 'N') OR (ANS = 'Y');
END;     (* PROCEDURE SHOW_SETTINGS *)

(*******************************************************)
(*                                                     *)
(*               FUNCTION NEXTDAY                      *)
(*                                                     *)
(*  Increments the date passed to it by one day.       *)
(*                                                     *)
(*******************************************************)
FUNCTION NEXTDAY (DATE : DATEREC) : DATEREC;
VAR
    DAYSINMONTH : 28..31;

BEGIN
CASE DATE.MONTH OF
   2          : DAYSINMONTH := 28;
   4,6,9,11   : DAYSINMONTH := 30;
    OTHERWISE DAYSINMONTH := 31;
   END;
DATE.DAY := (DATE.DAY MOD DAYSINMONTH) +1;  { next day }
IF DATE.DAY = 1 THEN
    BEGIN          { if next month }
    DATE.MONTH := (DATE.MONTH MOD 12) + 1;
    IF DATE.MONTH = 1 THEN   { if next year }
        DATE.YEAR := DATE.YEAR + 1;
    END;
```

```
NEXTDAY := DATE;        { return new date }
END;    (* FUNCTION NEXTDAY *)

(******************************************************)
(*                                                    *)
(*                FUNCTION TIMEDATE                   *)
(*                                                    *)
(*  Determines if system time and date are equal to   *)
(*  or after the time and date passed into function.  *)
(*  If so, function returns true.  Otherwise false is *)
(*  returned.                                         *)
(*                                                    *)
(******************************************************)
FUNCTION TIMEDATE (TIME : INTEGER; DATE : DATEREC) : BOOLEAN;
VAR
    SDATE : DATEREC;
    SDAYS, DAYS : INTEGER;
BEGIN
TIMEDATE := FALSE;
SYSDATE( SDATE );
SDAYS := SDATE.YEAR*372 + SDATE.MONTH*31 + SDATE.DAY;
DAYS := DATE.YEAR*372 + DATE.MONTH*31 + DATE.DAY;
IF SDAYS > DAYS THEN
    TIMEDATE := TRUE
ELSE
    BEGIN
    IF SDAYS = DAYS THEN
        IF SYSCLOCK > TIME THEN TIMEDATE := TRUE;
    END;
END;   (* FUNCTION TIMEDATE *)

(******************************************************)
(*                                                    *)
(*                  PROCEDURE PAUSE                   *)
(*                                                    *)
(*  Pauses program and displays current time and next *)
(*  starting time.                                    *)
(*                                                    *)
(*****************************()***********************)
PROCEDURE PAUSE (NEXTONTIME : INTEGER; NEXTSTARTDATE : DATEREC);
VAR
  I             : INTEGER;
  CURRENT       : TIMEREC;
  TIME          : TIMEREC;
BEGIN
PAGE;
FOR I := 1 TO 10 DO
  WRITELN ;
WRITELN ('WAITING FOR NEXT DATA COLLECTION PHASE');
CONVERT_TIMES( NEXTONTIME,RIGHT,TIME );
WRITELN('START AT      ',TIME.HOUR:2,':',TIME.MINUTE:2,':',
           TIME.CENTISECOND DIV 100 :2);
REPEAT     { wait for designated time and date }
  SYSTIME (CURRENT);
  WRITELN('IT IS NOW    ',CURRENT.HOUR:2,':',CURRENT.MINUTE:2,':',
           CURRENT.CENTISECOND DIV 100 :2);
  WRITE(CHR(31));          {moves cursor up one position}
  FOR I:=1 TO 100000 DO BEGIN END;
UNTIL TIMEDATE( NEXTONTIME, NEXTSTARTDATE );  {wait till time}
END;    (* PROCEDURE PAUSE *)

(*****************************************************)
(*                                                   *)
(*               PROCEDURE STOREHEADER               *)
(*                                                   *)
(*  Writes a header to disk file. Note that for      *)
(*  filename to be a unique name, sum of onduration  *)
(*  and off duration should be greater than one hour. *)
(*                                                   *)
(*****************************************************)
PROCEDURE STOREHEADER (VAR F : TEXT;
                       NEXTSTARTDATE : DATEREC;
                       NEXTONTIME,
                       ONDURATION : INTEGER;
                       SAMPLERATE,
                       ROLLAMPLITUDE,
                       ROLLPERIOD : REAL);
VAR
    TIME : TIMEREC;
    FILENAME : STRING [20];
    T : INTEGER;
BEGIN
FILENAME := '#4:B';
STRWRITE (FILENAME,5,T,NEXTSTARTDATE.MONTH :0);
FILENAME := FILENAME + '_';
STRWRITE (FILENAME,T+1,T,NEXTSTARTDATE.DAY :0);
FILENAME := FILENAME + 'H';
STRWRITE (FILENAME,T+1,T,NEXTONTIME DIV 360000 :0);
FILENAME := FILENAME + '.TEXT';
REWRITE (F, FILENAME);
WRITE (F,'DATE (MM/DD/YY) = ',NEXTSTARTDATE.MONTH:0);
WRITE (F,'/',NEXTSTARTDATE.DAY:0);
WRITELN (F,'/',NEXTSTARTDATE.YEAR:0);
WRITELN(PTR,'START DATE : ',NEXTSTARTDATE.MONTH:2,'/',NEXTSTARTDATE.DAY:2,
              '/',NEXTSTARTDATE.YEAR:2);
WRITELN (F,'START TIME (csec) = ',NEXTONTIME :0);
CONVERT_TIMES(NEXTONTIME,RIGHT,TIME);
WRITELN(PTR,'START TIME : ',TIME.HOUR:2,':',TIME.MINUTE:2,':',
                            TIME.CENTISECOND DIV 100 :2);
WRITELN (F,'ON DURATION (csec) = ',ONDURATION:0);
WRITELN(PTR,'DATA BLOCK LENGTH : ',ONDURATION DIV 6000 :4,' MINUTES');
WRITELN (F,'SAMPLE RATE = ',SAMPLERATE);   { rate buoy angle is recorded }
WRITELN(PTR,'SAMPLE RATE : ',SAMPLERATE:4,' Hz');
WRITELN (F,'ROLL AMPLITUDE = ',ROLLAMPLITUDE);
WRITELN (F,'ROLL PERIOD = ',ROLLPERIOD);
END;   { STOREHEADER }

(******************************************************)
(*                                                    *)
(*                  PROCEDURE PLOTANGLE               *)
(*                                                    *)
(*  Plots a line in window of CRT.  If right hand side *)
(*  of CRT is met then screen is cleared and pen moved *)
(*  to left margin of screen.                          *)
(*                                                     *)
(*******************************************************)
PROCEDURE PLOTANGLE (ANGLE : REAL; VAR TIME : INTEGER; DTIME : INTEGER);
BEGIN
IF TIME >= 3000 THEN
    BEGIN
    TIME := 0;               { set for left margin }
    MOVE (TIME,ANGLE);         { move pen to left margin }
    CLEAR_DISPLAY;              { clear graphics display }
    END
ELSE
    BEGIN      { plot line }
    TIME := TIME + DTIME;
    LINE (TIME,ANGLE);                    { draw line }
    END;
END;    (* PROCEDURE PLOTANGLE *)

(*******************************************************)
(*                                                     *)
(*                 GRAPHICS PLOT_AXES                  *)
(*                                                     *)
(*  Initiates graphics, sets viewport.  Draws axes     *)
(*  for buoy angle versus time plot.  Axes is drawn    *)
(*  on alpha screen, plotting is done on graphics      *)
(*  screen.  This saves time in clearing screen.       *)
(*                                                     *)
(*******************************************************)
PROCEDURE PLOT_AXES( MAXANGLE : REAL );
VAR
  I      : INTEGER;
  DY     : REAL;
BEGIN
PAGE;           { clear alpha screen }
WRITELN;
SET_ASPECT (511,389);   { set graphics screen }
SET_VIEWPORT (0.105,0.930,0.164,0.692);
SET_WINDOW (0,3000,-MAXANGLE,MAXANGLE);
DY := MAXANGLE / 4.5;
FOR I := 4 DOWNTO 1 DO BEGIN    { display upper y axes }
  WRITELN('    -|');
  WRITELN (I*DY:3:0,'--|');
END;
WRITELN('    -|');
WRITE (0:3,'--|');        { display zero degree line }
FOR I := 1 TO 71 DO
  WRITE ('-');
WRITELN;
FOR I := 1 TO 4 DO BEGIN        { display lower y axes }
  WRITELN('    -|');
  WRITELN (-I*DY:3:0,'--|');
END;
WRITE('    -|');          { display x axes }
FOR I := 1 TO 71 DO
  WRITE ('_');
```

```
WRITELN;
FOR I := 1 TO 11 DO             { ticks on x axes }
  WRITE ('        ^');
  WRITELN;
FOR I := 0 TO 10 DO             { number x axes }
  WRITE (I*3:7);
END;    (* GRAPHICS PROCEDURE PLOT_AXES *)

(*****************************************************)
(*                                                   *)
(*                 PROCEDURE TAKEDATA                *)
(*                                                   *)
(*  Records time interval on HP5334A and converts    *)
(*  to buoy angle.  Readings taken at regular        *)
(*  intervals specified by sample frequency.  Sends  *)
(*  readings to silo.  If sample freq. 3Hz or less   *)
(*  then also records waverider receiver output volts *)
(*  Plots buoy angle on CRT in real time.            *)
(*                                                   *)
(*****************************************************)
PROCEDURE TAKEDATA (VAR F : TEXT;
                    ONDURATION : INTEGER;
                    SAMPLERATE,
                    SIGFREQ,
                    OFFSET : REAL);
{Note: for this to work properly, there must exist a 90-degree phase shift
between signal and reference wave when buoy is vertical. Rotate gear on
motor shaft to adjust phase shift.}
VAR
    B             : FILE OF REAL;
    W             : TEXT;
    I, T          : INTEGER;     { counter }
    DATA          : STRING[21];       { input buffer }
    BUOYANGLE     : REAL;
    BUOYANG2      : REAL;
    WAVEHEIGHT    : REAL;
    SAMPTIME      : INTEGER;    { time in centisecs between readings }
    NUMREADINGS   : INTEGER;     { counter of readings taken }
    INCLUDEWAVE   : BOOLEAN;     { true if waverider signal is recorded }
    NEWDATA       : BOOLEAN;
    OVERFLOW      : BOOLEAN;
    ERRORRETURN   : INTEGER;       {variable for initialization outcome}
    TIME          : INTEGER;     {time in centiseconds from plot origin}
    TDATA         : TIMERDATA;
    OLDISRHOOK    : KBDHOOKTYPE;

{************************************************************************}
PROCEDURE GETREADING (VAR STATBYTE, DATABYTE : BYTE; VAR DOIT : BOOLEAN);
{this procedure uses global data from TAKEDATA}
BEGIN
IF OVERFLOW THEN
    BEGIN
    WRITELN('DIGITIZING FREQUENCY TOO HIGH');
    IOCONTROL (GPIO,3,CLOSE_LENS);  { close lens and shut off engine }
    CLOSE(F);  {close without saving}
    END;
```

```
OVERFLOW := TRUE;  {if this is not reset to false, proc never finished}
READSTRING( HP5334A, DATA );
STRREAD( DATA,2,T,BUOYANGLE );
READSTRING( HP5334A, DATA );
STRREAD( DATA,2,T,BUOYANG2 );
BUOYANGLE := (BUOYANGLE + BUOYANG2) / 2;     {average out some of noise}
BUOYANGLE := (BUOYANGLE - OFFSET) * SIGFREQ * 180;
WRITE( B, BUOYANGLE );
IF INCLUDEWAVE THEN
    BEGIN
    READSTRING( K485, DATA );
    WRITELN (W, DATA);
    END;
NEWDATA := TRUE;
OVERFLOW := FALSE;
END;
{**************************************************************************}
BEGIN   {TAKEDATA}
{REWRITE (B, 'RAM:BDATA');
REWRITE (W, 'RAM:WDATA');}   REWRITE (B);  REWRITE (W);
IOCONTROL (GPIO,3,OPEN_LENS);   { open lens, start engine }
FOR I:=1 TO 200 DO READSTRING( HP5334A, DATA );  { waits about 20 sec }
GRAPHICS_INIT;
DISPLAY_INIT(CRTADDR,CONTROLWORD,ERRORRETURN);
IF ERRORRETURN <> 0 THEN HALT;  { if error halt }
PLOT_AXES(MAXANGLE);              { plot graph axes on alpha screen }
TIME := 3000;
NUMREADINGS := ROUND((ONDURATION DIV 100) * SAMPLERATE);
SAMPTIME := ROUND (100 / SAMPLERATE);         {time in csec between samples}
IF SAMPLERATE <= 3 THEN INCLUDEWAVE := TRUE
    ELSE INCLUDEWAVE := FALSE;
PLOTANGLE (0.0,TIME,SAMPTIME);   { place pen }
OVERFLOW := FALSE;
NEWDATA := FALSE;
BUOYANGLE := 0;

TRY
    OLDISRHOOK := TIMERISRHOOK;   {save old ISR to later restore}
    TIMERISRHOOK := GETREADING;   {assign new ISR}
    TDATA.COUNT := SAMPTIME;
    CALL( TIMERIOHOOK, CYCLICT, SETT, TDATA );    {set cyclic timer}
    CALL( MASKOPSHOOK, TIMERMASK, 0 );  {enable timer}

    I := 0;
    REPEAT
        IF NEWDATA THEN
            BEGIN
            I := I + 1;
            NEWDATA := FALSE;
            PLOTANGLE (BUOYANGLE,TIME,SAMPTIME);
            END;
    UNTIL I >= NUMREADINGS;
    ESCAPE (0);               {generate an error}

RECOVER
    BEGIN
    CALL( MASKOPSHOOK, 0, TIMERMASK );      {disable timer isr}
    TDATA.COUNT := 0;
    CALL( TIMERIOHOOK, CYCLICT, SETT, TDATA );     {zero timer}
    TIMERISRHOOK := OLDISRHOOK;                 {restore old ISR}
    END;

CLEAR_DISPLAY;            { clearm graphics from CRT }
GRAPHICS_TERM;            {terminate the graphics package}
IOCONTROL (GPIO,3,CLOSE_LENS);  { close lens and shut off engine }

RESET (B);                      {transfer data to disk}
RESET (W);
WHILE NOT EOF (B) DO
    BEGIN
    READ (B, BUOYANGLE);
    WRITELN( F, BUOYANGLE );
    IF INCLUDEWAVE THEN
        BEGIN
        READLN (W, DATA);
        STRREAD( DATA,5,T,WAVEHEIGHT );
        WAVEHEIGHT := WAVEHEIGHT * WAVECAL;
        WRITELN( F,'W',WAVEHEIGHT );
        END;
    END;
CLOSE (B, 'SAVE');
CLOSE (W, 'SAVE');

END;   (* PROCEDURE TAKEDATA *)

(*****************************************************)
(*                                                   *)
(*                  MAIN PROGRAM                     *)
(*                                                   *)
(*****************************************************)
BEGIN
WRITELN ('HAVE YOU INSTALLED A RAM VOLUME NAME "RAM:"? (Y) or (N)');
READ (ANS);
IF (ANS = 'N') OR (ANS = 'n') THEN HALT;
WRITELN;
REWRITE(PTR,'PRINTER:');
IOINITIALIZE;                   {reset all interfaces}
IOCONTROL(GPIO,3,CLOSE_LENS);   {insure lens is closed first thing}
SYSDATE (TODAYSDATE);     { set current date to system date }
NEXTSTARTDATE := TODAYSDATE;  { default settings }
FINALSTOPDATE := TODAYSDATE;
STARTTIME := 7560000;     {21:00:00}
STOPTIME := 8280000;      {23:00:00}
ONDURATION := 180000;     {00:30:00}
OFFDURATION := 360000;    {01:00:00}
SAMPLERATE := 5;
OFFSET := 16.0E-3;
SIGNALFREQ := 31.5;
REPEAT            { get user provided inputs }
    SET_DATES( TODAYSDATE, NEXTSTARTDATE, FINALSTOPDATE);
    SET_TIMES( STARTTIME, STOPTIME, ONDURATION, OFFDURATION);
    SET_DATARATES (SAMPLERATE, OFFSET, SIGNALFREQ);
    WRITELN ('INPUT BUOY TYPE (8x26, 5x11, etc.)' );
    READLN (BUOYTYPE);
    SHOW_SETTINGS (ANS, TODAYSDATE, NEXTSTARTDATE, FINALSTOPDATE,
                      STARTTIME, STOPTIME,
                      ONDURATION, OFFDURATION,
                      SAMPLERATE, OFFSET, SIGNALFREQ);
UNTIL ANS = 'N';
WRITELN;
WRITELN ('****INSERT DATA OUTPUT DISK IN RIGHT HAND DRIVE****');
READ (ANS);
NEXTONTIME := STARTTIME;
IF STARTTIME>STOPTIME THEN          {crosses midnight}
    NEXTSTOPDATE := NEXTDAY( NEXTSTARTDATE )
ELSE
    NEXTSTOPDATE := NEXTSTARTDATE;
REPEAT          { go until final stop date and time }
    PAUSE ( NEXTONTIME, NEXTSTARTDATE );   { wait till collection phase }
    STOREHEADER (F,NEXTSTARTDATE,NEXTONTIME,ONDURATION,SAMPLERATE,0,0);
    WRITELN (F,BUOYTYPE,' BUOY ANGLE VS TIME');
    TAKEDATA (F,ONDURATION,SAMPLERATE,SIGNALFREQ,OFFSET);
    CLOSE (F,'SAVE');
    WRITELN(PTR,'END OF DATA ACQUISITION');
    WRITELN(PTR);
    WRITELN(PTR);
    NEXTONTIME := (NEXTONTIME + ONDURATION + OFFDURATION) MOD MIDNIGHT;
    SYSDATE (TODAYSDATE);   {update todaysdate}
    IF NEXTONTIME < SYSCLOCK THEN        {crosses midnight}
        NEXTSTARTDATE := NEXTDAY( TODAYSDATE )
    ELSE                                 {do not cross midnight yet}
        NEXTSTARTDATE := TODAYSDATE;
    IF TIMEDATE( STOPTIME, NEXTSTOPDATE ) THEN   {end of nights collection}
        BEGIN
        NEXTONTIME := STARTTIME;
        IF STOPTIME > STARTTIME THEN     {collection does not cross midnight}
            BEGIN
            NEXTSTARTDATE := NEXTDAY( NEXTSTOPDATE );
            NEXTSTOPDATE := NEXTSTARTDATE;
            END
        ELSE                             {collection does cross midnight}
            NEXTSTOPDATE := NEXTDAY( NEXTSTARTDATE );
        END;
UNTIL TIMEDATE( STOPTIME, FINALSTOPDATE );
PAGE;          { clear screen }
FOR I := 1 TO 10 DO     { terminate message }
  WRITELN;
SYSTIME (TIME);
WRITE ('Program Terminated at ',TIME.HOUR:2,':');
WRITELN (TIME.MINUTE:2,':',TIME.CENTISECOND DIV 100:2);
END.    (* PROGRAM BUOY_MOTION *)
```

Program **BUOY_PROBABILITIES**

```
PROGRAM BUOY_PROBABILITIES (INPUT, OUTPUT);
{Determines the probabilities of detection and recognition of flashing lights
on rolling buoys.  Input data is in user specified file on left disk.  Data
was stored using either PROGRAM BUOY_MOTION or PROGRAM ROLLSIM & must be on
left disk. Output stored on right disk (#4:).  LANT.REAL must be on left (#3:)
disk. LANT.REAL was generated with TRANSLAMP.CODE.  Before executing program
operator (or Boot disk) must set up a ram volume named RAM: .
By Dan Brown, Apr 1987.}

IMPORT
    RND,
    IODECLARATIONS,
    SYSDEVS,
    DGL_LIB,
    DGL_INQ;

CONST
    BUOYRATE         = 6;   {maximum buoy angle sample rate}
    NUMANGLES        = 455; {number inten vs angle data points in LANTARRAY}
    RAMFILE          = 'RAM:DATA.REAL';   {file name of inten vs time}
    NUMDISTS         = 16;   {number of different distances from light}
    NUMTIMES         = 15;  {number of different observation times}
    OBSTIMEINC       = 2;   {integer no. seconds per time increment}
    RATEMULTIPLIER   = 7; {no. sample points multiplied for better resolution}
    WINDOWLENGTH     = OBSTIMEINC * NUMTIMES * RATEMULTIPLIER * BUOYRATE;
    NUMWINDOWS       = 500;  {number data windows, NUMTIMES wide, analyzed}
    ESC              = CHR(27);

TYPE
    CHARACTERISTICS = (FL6, FL4, FL25, QKFL, GPFL5, GPFL6, IQKFL, MOA,
                       EINT6, OCC4);       {the ten standard flash
                       characteristics listed on page 6-52 of AtoN Technical}
    MSG_TYPE       = STRING[255];
    DIR_TYPE       = ( RIGHT, LEFT );      {enumerated direction type}
    PROBRECORD = RECORD
                   POD : ARRAY [1..NUMDISTS,1..NUMTIMES] OF REAL;
                   POR : ARRAY [1..NUMDISTS,1..NUMTIMES] OF REAL;
                   EFI  : ARRAY [1..NUMDISTS] OF REAL;
                   DISTANCEINC : REAL;
                   TIMEINC : REAL;
                 END;

    PERRECORD  = RECORD
                   DETECDIST : REAL;
                   RECOGDIST : REAL;
                 END;
    PERIODINFO = ARRAY[1..NUMTIMES] OF PERRECORD;   {gives detection
                                                      and recognition distance
                                                      of each flash period in
                                                      a window of data}
    HEADRECORD = RECORD
                   DAY   : 1..31;
                   MONTH : 1..12;
                   YEAR  : 86..90;
                   STARTTIME : INTEGER;   {time in centiseconds past midnight}
```

```
                    ONDURATION : INTEGER;  {data block length in centiseconds}
                    SAMPLERATE : REAL;     {rate in Herz}
                    ROLLAMPLITUDE : REAL;  {amplitude in degrees off vertical}
                    MEANROLLANGLE : REAL;       {expected value of tilt angle}
                    ROLLPERIOD : REAL;     {period in seconds}
                  END;
    LANTRECORD = RECORD
                    ANGLE     : REAL;
                    INTENSITY : REAL;
                  END;
    LANTARRAY = ARRAY [Ø..NUMANGLES] OF LANTRECORD;  {array of intensity vs
                                                      angle from user specified
                                                      lantern file}
    FILENAMETYPE = STRING [20];
    FILENAMEARRAY = ARRAY [1..10] OF FILENAMETYPE; {array of disk locations
                                                    of data sets to be analyzed}
    DATAWINDOWS = RECORD
                     INTENSITY : ARRAY [Ø..WINDOWLENGTH] OF REAL;
                     SAMPLERATE : REAL;
                     MAXEFI : ARRAY [1..NUMTIMES] OF REAL; {detection EFI}
                     MINEFI : ARRAY [1..NUMTIMES] OF REAL; {recognition EFI}
                   END;
    PROBFILE = FILE OF PROBRECORD;
    FLASHARRAY = ARRAY [Ø..100 * RATEMULTIPLIER] OF Ø..1;

VAR
    I             : INTEGER;
    PROBABILITY   : PROBRECORD;
    HEADER        : HEADRECORD;
    FILENAMES     : FILENAMEARRAY;
    LANTERNFILE   : FILENAMETYPE;   {file name of inten vs angle}
    MINUTES       : INTEGER;       {number of minutes of data set to analyze}
    FLASHCHAR     : CHARACTERISTICS;   {flash characteristic being analyzed}
    OUTDISK       : PROBFILE;
          {following variables refer to intensity record on ram disk}
    FILEPOS       : INTEGER;  {file position (in ram file) of data window}
    SEED          : INTEGER;    {dummy variable for random number generator}
    RANGE         : INTEGER;    {range of random numbers for file position}
    NUMREADINGS   : INTEGER;    {number of data points stored on ram file}
          {following variables refer to WINDOW.INTENSITY array}
    WINDOW        : DATAWINDOWS;    {15 periods of intensity vs time data}
    FLASH         : FLASHARRAY;  {normalized flash period}
    FLASHLEN      : INTEGER;    {number of data points in period in FLASH}
    W             : 1..NUMWINDOWS;   {counter for number of windows analyzed}
    STARTPOS      : INTEGER;   {start position in FLASH for normalizing win.}
    TIME          : 1..NUMTIMES;  {counter to zero arrays}
    PRIN          : TEXT;
{*****************************************************************************}
{*****************************************************************************}
{*****************************************************************************}
PROCEDURE PLOT_WINDOW (VAR W : DATAWINDOWS);
{Plots data array pointed to by P. Used for testing.}
VAR
    I : INTEGER;              {loop counter}
    CHARWIDTH, CHARHEIGHT : REAL;
    XMIN, XMAX, YMIN, YMAX, X, Y : REAL;  {x and y plot values}
    FIRSTPOINT : BOOLEAN;
    MAXMAG : REAL;            {maximum magnitude in data}
{*************************************************************************}
PROCEDURE INITIALIZE_PLOT;
CONST
    CRTADDR = 3;  {3=CRT, 705=PLOTTER}
    PLTADDR = 705;
    CONTROLWORD = 0;
    GET_ASPECT=254;
VAR
    ERROR : INTEGER;
    DUMMY : INTEGER;
    RATIO_LIST : ARRAY[1..2] OF REAL;
BEGIN
GRAPHICS_INIT;
DISPLAY_INIT(CRTADDR,CONTROLWORD,ERROR);
INQ_WS(GET_ASPECT,0,0,2,DUMMY,DUMMY,RATIO_LIST,ERROR);
IF ERROR=0 THEN SET_ASPECT(1.0,RATIO_LIST[2]);
{SET_VIEWPORT(0.0,1.00,0.0,0.74);}      {Sets plotting area of CRT}
SET_VIEWPORT(0.07,1.00,0.04,0.7);     {Sets plotting area of CRT}
END;
{*************************************************************************}
PROCEDURE SET_USER_SCALE (VAR W    : DATAWINDOWS;
                          VAR XMIN,
                              XMAX,
                              YMIN,
                              YMAX,
                              CHARWIDTH,
                              CHARHEIGHT : REAL);
VAR
    VALUE, MAXVALUE : REAL;
    I : INTEGER;
BEGIN
MAXVALUE := 0;
WITH W DO
    FOR I:=0 TO WINDOWLENGTH DO         {find maximum value in data array}
        BEGIN
        VALUE := INTENSITY [I];
        IF VALUE>MAXVALUE THEN MAXVALUE:=VALUE;
        END;
XMIN:=0.0;
XMAX:= 30;         {number of seconds}
YMIN:= 0;
YMAX:= MAXVALUE;
SET_WINDOW(XMIN,XMAX,YMIN,YMAX);       {Sets user coord. scale}
CHARWIDTH := (XMAX - XMIN) / 50;
CHARHEIGHT := (YMAX - YMIN) / 25;
SET_CHAR_SIZE(CHARWIDTH,CHARHEIGHT);   {sets constant relative size}
END;
{*************************************************************************}
PROCEDURE LABEL_PLOT ( XMIN,
                       XMAX,
                       YMIN,
                       YMAX,
                       CHARWIDTH,
                       CHARHEIGHT : REAL );
```

```
VAR
    TITLE : STRING[80];
    X, DX, Y, DY : REAL;
    I, J, K : INTEGER;
BEGIN
MOVE(XMAX,0);
LINE(0,0);
X := 0;
DX := 1.0;             {Label x axis}
WHILE X < XMAX DO      {major tick marks, horizontal axis}
    BEGIN
    X := X + DX;
    MOVE(X,0);
    LINE(X,-CHARHEIGHT);
    TITLE:='';                      {label major ticks}
    STRWRITE(TITLE,1,K,X:0:0);
    MOVE(X-STRLEN(TITLE)*CHARWIDTH+CHARWIDTH/2,-1.2*CHARHEIGHT);
    GTEXT(TITLE)
    END;
MOVE(0,YMAX);
LINE(0,YMIN);
LINE(-CHARWIDTH,YMIN);
Y := YMIN;
DY := (YMAX - YMIN)/10;         {vertical axis tic marks}
FOR I:=1 TO 10 DO
    BEGIN
    Y := Y + DY;
    MOVE(0,Y);
    LINE(-0.5 * CHARWIDTH,Y);
    END;
TITLE:='';
STRWRITE(TITLE,1,K,YMAX:4:1);
MOVE(-STRLEN(TITLE)*CHARWIDTH,YMAX);
GTEXT(TITLE);
TITLE := 'Data Window';
MOVE ( (XMAX - XMIN - STRLEN(TITLE)*CHARWIDTH)/3 + XMIN, YMAX );
GTEXT(TITLE);
END;
{***************************************************************************}
BEGIN   {PLOT_DATA}
INITIALIZE_PLOT;
SET_USER_SCALE (W,XMIN,XMAX,YMIN,YMAX,CHARWIDTH,CHARHEIGHT);
LABEL_PLOT (XMIN,XMAX,YMIN,YMAX,CHARWIDTH,CHARHEIGHT);
MOVE(0, W.INTENSITY [0]);
I := 0;
WHILE X < 30.0 DO
    BEGIN
    I := I + 1;
    X := I / W.SAMPLERATE;
    LINE(X,W.INTENSITY [I]);
    END;
CALL(DUMPGRAPHICSHOOK);
CLEAR_DISPLAY;
GRAPHICS_TERM;
WRITELN (PRIN, 'MAXEFI                                 MINEFI');
FOR I:=1 TO NUMTIMES DO
```

```
    WRITELN (PRIN, W.MAXEFI [I] :0:2, W.MINEFI [I] :40:2);
PAGE (PRIN);
END;
{****************************************************************************}
{****************************************************************************}
{****************************************************************************}
PROCEDURE INPUT_USER_DATA (VAR FILENAMES : FILENAMEARRAY;
                           VAR LANTFILE : FILENAMETYPE);
{FILENAMES specifies data filenames to be analyzed, LANTFILE specifies the
defined lantern type (vertical intensity profile) to use.}
VAR
    NUMSETS : 1..10;
    I, J : INTEGER;
    C : CHAR;
BEGIN
FOR I:=1 TO 10 DO WRITELN;
WRITELN ('HAVE YOU INSTALLED A RAM VOLUME NAMED, "RAM" ? (Y) or (N)');
READ (C);
IF (C = 'N') OR (C='n') THEN HALT;
PAGE;
FOR I:=1 TO 10 DO WRITELN;
WRITELN('INPUT NAME OF LANTERN FILE (excluding volume no.)');
READLN (LANTFILE);
LANTFILE := '#3:' + LANTFILE;
PAGE;
FOR I:=1 TO 10 DO WRITELN;
WRITELN('INPUT NUMBER OF SETS OF DATA TO ANALYZED (1..10)');
READLN (NUMSETS);
FOR J:= 1 TO NUMSETS DO
    BEGIN
    PAGE;
    FOR I:=1 TO 10 DO WRITELN;
    WRITELN('INPUT DATA FILE NAME (excluding vol. no.) FOR RUN NUMBER ',J);
    READLN( FILENAMES [J] );
    FILENAMES [J] := '#3:' + FILENAMES [J];
    END;
FOR J:= NUMSETS + 1 TO 10 DO
    FILENAMES [J] := 'NONE';    {marks end of sets}
PAGE;
FOR I:=1 TO 10 DO WRITELN;
WRITELN (' ':22, 'PLACE DATA INPUT DISK IN LEFT HAND DRIVE....(#3)');
WRITELN (' ':22, 'THIS DISK MUST ALSO HAVE LANTERN FILE ON IT');
WRITELN (' ':22, 'PLACE OUTPUT DISC IN RIGHT HAND DRIVE....(#4)');
WRITELN;
WRITE (' ':22, 'TYPE ANY CHARACTER WHEN READY...  ');
READ (C);
WRITELN;
END;
{****************************************************************************}
{****************************************************************************}
{****************************************************************************}
PROCEDURE INITPROB_RECORD (VAR PROBABILITY : PROBRECORD);
{sets all probabilities to zero, sets the observer distance and effective
intensity fields}
VAR
    DIST : INTEGER;
    OBSTIME : INTEGER;
BEGIN
WITH PROBABILITY DO
    BEGIN
    DISTANCEINC := 0.5;
    TIMEINC := OBSTIMEINC;
    EFI [1] := 0.195;          {0.5 n.miles}
    EFI [2] := 0.905;          {1.0 n.miles}
    EFI [3] := 2.368;          {1.5 n.miles}
    EFI [4] := 4.894;          {2.0 n.miles}
    EFI [5] := 8.889;          {2.5 n.miles}
    EFI [6] := 14.881;          {3.0 n.miles}
    EFI [7] := 23.545;          {3.5 n.miles}
    EFI [8] := 35.749;          {4.0 n.miles}
    EFI [9] := 54.597;          {4.5 n.miles}
    EFI [10] := 75.484;          {5.0 n.miles}
    EFI [11] := 106.176;          {5.5 n.miles}
    EFI [12] := 146.888;          {6.0 n.miles}
    EFI [13] := 200.399;          {6.5 n.miles}
    EFI [14] := 270.177;          {7.0 n.miles}
    EFI [15] := 360.545;          {7.5 n.miles}
    EFI [16] := 470.871;          {8.0 n.miles}
    FOR DIST:=1 TO NUMDISTS DO
        FOR OBSTIME:=1 TO NUMTIMES DO
            BEGIN
            POD [DIST,OBSTIME] := 0;
            POR [DIST,OBSTIME] := 0;
            END;
    END;

END;
{***************************************************************************}
{***************************************************************************}
{***************************************************************************}
PROCEDURE STOREDATA_IN_RAMFILE (FILENAME : FILENAMETYPE;
                                LANTERNFILE : FILENAMETYPE;
                            VAR HEADER : HEADRECORD;
                            VAR READINGS : INTEGER);
{reads disk data input, converts buoy versus time data into intensity
versus time data for a fixed on light.  FILENAME is data location on disk.
LANTERNFILE is specified file name of intensity vs angle data on floppy disk.
If data is actual field data then program will print RMS roll
amplitude. If data simulated (with ROLLSIM) then peak roll amplitude and
roll period are printed out. Actual vs simulated is determined by whether or
not HEADER.ROLLPERIOD is nonzero. Returns number of READINGS stored on ram
disk.  Multiplies number of data points by RATEMULTIPLIER for greater
resolution when determining effective intensities.}
VAR
    C : CHAR;
    T, I : INTEGER;
    NUM : REAL;
    LANTERNDATA : LANTARRAY;
    DATASIMULATED : BOOLEAN;  {distinguish between actual and simulated}
    WAVEINCLUDED : BOOLEAN; {true if wave rider data included}
    RMSROLL : REAL;     {root mean square roll amplitude, for actual data}
    MEANROLL : REAL;
    TEMP : ARRAY [0..RATEMULTIPLIER] OF REAL;  {temporary array of buoy data}
    ANGLEINC : REAL;
    F : FILE OF REAL; {ram disk of intensity vs time}
    PRIN : TEXT;      {printer}
    DSK : TEXT;       {floppy disk of input data}
    STR : STRING [40];   {variable to read DSK}
{****************************************************************************}
PROCEDURE GET_LANTERN_DATA (VAR LANTERNDATA : LANTARRAY;
                            VAR LANTERNFILE : FILENAMETYPE);
{reads intensity versus angle data from lantern file stored on floppy disk
specified in CONST block. Data used to convert angle vs time data on silo to
intensity vs time which is stored on ram disk. If different lantern file is
needed, must change file name in main CONST block.}
VAR
    F : FILE OF REAL;
    I : INTEGER;
BEGIN
RESET (F, LANTERNFILE);
LANTERNDATA [0].ANGLE := -95.0;      {assures that up to 90-deg rolls can}
LANTERNDATA [0].INTENSITY := 0.0;    {be converted to intensity values}
I:=1;
WHILE NOT EOF(F) DO
    WITH LANTERNDATA [I] DO
        BEGIN
        READ (F, ANGLE, INTENSITY);
        I:=I+1;
        END;
CLOSE (F);
WHILE I<=NUMANGLES DO
    WITH LANTERNDATA [I] DO
        BEGIN
        ANGLE := 95;
        INTENSITY := 0;
        I:=I+1;
        END;
END;
{****************************************************************************}
FUNCTION ANGLE_TO_INTENSITY (VAR LANTERNDATA : LANTARRAY;
                                 BUOYANGLE : REAL) : REAL;
{converts buoy angle to lantern intensity.  max angle ever seen by this
function is + or - 90 degrees due to data acquisition algorithm}
VAR
    I : INTEGER;
    I1, A1 : REAL;
BEGIN
I:=1;
WHILE BUOYANGLE > LANTERNDATA [I].ANGLE DO I:=I+1;
WITH LANTERNDATA [I-1] DO
    BEGIN
    I1 := INTENSITY;
    A1 := ANGLE;
    END;
WITH LANTERNDATA [I] DO
    ANGLE_TO_INTENSITY := I1+(INTENSITY-I1)*(BUOYANGLE-A1)/(ANGLE-A1);
END;
{****************************************************************************}
```

```
BEGIN {STOREDATA_IN_RAMFILE}
GET_LANTERN_DATA (LANTERNDATA, LANTERNFILE);  {get intensity vs angle data}
                {reset devices}
RESET (DSK, FILENAME);
REWRITE (PRIN, 'PRINTER:');
OPEN (F, RAMFILE);      {open ram file for writing to}
CLOSE (F, 'PURGE');        {purges old file if it exists}
REWRITE (F, RAMFILE);      {open ram file for writing to}
                {read file header}
READLN (DSK, STR);
WRITELN (PRIN, STR);
T:=1;
REPEAT
    STRREAD (STR,T,T,C);
UNTIL C='=';
STRREAD (STR,T,T,HEADER.MONTH);
STRREAD (STR,T+1,T,HEADER.DAY);
STRREAD (STR,T+1,T,HEADER.YEAR);
WRITELN (HEADER.MONTH,'/',HEADER.DAY,'/',HEADER.YEAR);
READLN (DSK, STR);
WRITELN (PRIN, STR);
T:=1;
REPEAT
    STRREAD (STR,T,T,C);
UNTIL C='=';
STRREAD (STR,T,T,HEADER.STARTTIME);
READLN (DSK, STR);
WRITELN (PRIN, STR);
T:=1;
REPEAT
    STRREAD (STR,T,T,C);
UNTIL C='=';
STRREAD (STR,T,T,HEADER.ONDURATION);
WRITELN('ON DURATION = ',HEADER.ONDURATION DIV 6000 :0, ' minutes');
WRITELN;
READLN (DSK, STR);
WRITELN (PRIN, STR);
T:=1;
REPEAT
    STRREAD (STR,T,T,C);
UNTIL C='=';
STRREAD (STR,T,T,HEADER.SAMPLERATE);
IF HEADER.SAMPLERATE > 3 THEN
    WAVEINCLUDED := FALSE
ELSE
    WAVEINCLUDED := TRUE;         {3 Hz or less includes wave rider}
READLN (DSK, STR);
WRITELN (PRIN, STR);
T:=1;
REPEAT
    STRREAD (STR,T,T,C);
UNTIL C='=';
STRREAD (STR,T,T,HEADER.ROLLAMPLITUDE);
READLN (DSK, STR);
WRITELN (PRIN, STR);
T:=1;
```

```
REPEAT
    STRREAD (STR,T,T,C);
UNTIL C='=';
STRREAD (STR,T,T,HEADER.ROLLPERIOD);
IF HEADER.ROLLPERIOD <> 0 THEN
    DATASIMULATED := TRUE
ELSE
    DATASIMULATED := FALSE;
READLN (DSK, STR);
WRITELN (PRIN, STR);
PAGE (PRIN);
                {read data and store on disk}
READLN (DSK, NUM);
WRITE (F, ANGLE_TO_INTENSITY (LANTERNDATA, NUM) );
TEMP [RATEMULTIPLIER] := NUM;
READINGS := 1;
IF NOT DATASIMULATED THEN
    BEGIN
    RMSROLL := NUM * NUM;
    MEANROLL := ABS (NUM);
    END
ELSE
    BEGIN
    RMSROLL := 0;
    MEANROLL := 0;
    END;
IF WAVEINCLUDED THEN      {throw away wave data}
    BEGIN
    READ (DSK, C);
    READLN (DSK, NUM);
    END;

WHILE NOT EOF(DSK) DO
    BEGIN
    READLN (DSK, NUM);
    READINGS := READINGS + RATEMULTIPLIER;
    IF NOT DATASIMULATED THEN
        BEGIN
        RMSROLL := RMSROLL + NUM*NUM;   {sum the squares}
        MEANROLL := MEANROLL + ABS( NUM );
        END;
    TEMP [0] := TEMP [RATEMULTIPLIER];
    TEMP [RATEMULTIPLIER] := NUM;
    ANGLEINC := (TEMP [RATEMULTIPLIER] - TEMP [0]) / RATEMULTIPLIER;
    FOR I := 1 TO RATEMULTIPLIER - 1 DO
        TEMP [I] := I * ANGLEINC + TEMP [0];
    FOR I := 1 TO RATEMULTIPLIER DO
        BEGIN
        NUM := ANGLE_TO_INTENSITY (LANTERNDATA, TEMP [I]);
        WRITE (F, NUM);
        END;
    IF WAVEINCLUDED THEN      {throw away wave data}
        BEGIN
        READ (DSK, C);
        READLN (DSK, NUM);
        END;
    END;

CLOSE (DSK);
CLOSE (PRIN);
CLOSE (F, 'SAVE');
IF NOT DATASIMULATED THEN
    BEGIN
    RMSROLL := SQRT( RMSROLL / (READINGS DIV RATEMULTIPLIER + 1) );
    HEADER.ROLLAMPLITUDE := RMSROLL;    {ROLLPERIOD still zero for actual}
    HEADER.MEANROLLANGLE := MEANROLL / (READINGS DIV RATEMULTIPLIER + 1);
    END;
WRITELN('FINISHED READING DATA FROM DISK');
END;
{****************************************************************************}
{****************************************************************************}
{****************************************************************************}
PROCEDURE GET_DATA_WINDOW (VAR WIN : DATAWINDOWS;  {window of inten/time}
                               FILEPOS : INTEGER);  {start position in file}
{Reads a window of data from ram file beginning at a position in file
indicated by FILEPOS. Length of window is max observation time.}
VAR
    F           : FILE OF REAL;
    INTENDIF    : REAL;    {inten diff between two actual sample points}
    NUMPOINTS   : INTEGER;  {number of data points to read into window}
    I, IL       : INTEGER;  {indices}
    DI          : 0..RATEMULTIPLIER;   {index difference}
BEGIN
OPEN (F, RAMFILE);   {point to start of data record}
NUMPOINTS := ROUND (OBSTIMEINC * NUMTIMES * WIN.SAMPLERATE);
READDIR (F, FILEPOS, WIN.INTENSITY [0]);  {positions to FILEPOS and reads}
I := 0;
WHILE I<NUMPOINTS DO
    BEGIN
    READ (F, WIN.INTENSITY [I]);
    I := I + 1;
    END;
WHILE I<WINDOWLENGTH DO
    BEGIN
    I := I + 1;
    WIN.INTENSITY [I] := 0;
    END;
CLOSE (F);
END;
{****************************************************************************}
{****************************************************************************}
{****************************************************************************}
PROCEDURE FLASH_PROPERTIES (VAR FLASH      : FLASHARRAY;
                            VAR PERLEN     : INTEGER;
                                FLASHCHAR  : CHARACTERISTICS;
                                RATE       : REAL);
{Inputs a normalized flash period in FLASH to be used for finding flashed
intensity vs time.  Samplerate in FLASH is same as WINDOW.INTENSITY, ie.
buoy roll samplerate times ratemultiplier.  Returns number of points per
flash period in PERLEN.}
VAR
    I, J, TURNOFF, TURNON : INTEGER;
```

```
BEGIN
FLASH [0] := 0;
I := 1;
CASE FLASHCHAR OF
    FL6   : BEGIN
            TURNOFF := ROUND ( 0.6 * RATE ) + 1;
            WHILE I < TURNOFF DO
                BEGIN
                FLASH [I] := 1;
                I := I + 1;
                END;
            PERLEN := ROUND(6 * RATE);
            END;
    FL4   : BEGIN
            TURNOFF := ROUND ( 0.4 * RATE ) + 1;
            WHILE I < TURNOFF DO
                BEGIN
                FLASH [I] := 1;
                I := I + 1;
                END;
            PERLEN := ROUND(4 * RATE);
            END;
    FL25  : BEGIN
            TURNOFF := ROUND ( 0.3 * RATE ) + 1;
            WHILE I < TURNOFF DO
                BEGIN
                FLASH [I] := 1;
                I := I + 1;
                END;
            PERLEN := ROUND(2.5 * RATE);
            END;
    QKFL  : BEGIN
            TURNOFF := ROUND ( 0.3 * RATE ) + 1;
            WHILE I < TURNOFF DO
                BEGIN
                FLASH [I] := 1;
                I := I + 1;
                END;
            PERLEN := ROUND(RATE);
            END;
    GPFL5 : BEGIN
            TURNOFF := ROUND (0.4 * RATE) + 1;
            WHILE I < TURNOFF DO
                BEGIN
                FLASH [I] := 1;
                I := I + 1;
                END;
            TURNON := ROUND (RATE) + 1;
            WHILE I < TURNON DO
                BEGIN
                FLASH [I] := 0;
                I := I + 1;
                END;
            TURNOFF := ROUND (1.4 * RATE) + 1;
            WHILE I < TURNOFF DO
                BEGIN
            FLASH [I] := 1;
            I := I + 1;
            END;
        PERLEN := ROUND(5 * RATE);
        END;
GPFL6 : BEGIN
        TURNOFF := ROUND (RATE) + 1;
        WHILE I < TURNOFF DO
            BEGIN
            FLASH [I] := 1;
            I := I + 1;
            END;
        TURNON := ROUND (2 * RATE) + 1;
        WHILE I < TURNON DO
            BEGIN
            FLASH [I] := Ø;
            I := I + 1;
            END;
        TURNOFF := ROUND (3 * RATE) + 1;
        WHILE I < TURNOFF DO
            BEGIN
            FLASH [I] := 1;
            I := I + 1;
            END;
        PERLEN := ROUND(6 * RATE);
        END;
IQKFL : BEGIN
        FOR J := 1 TO 6 DO
            BEGIN
            TURNOFF := ROUND( (J - 1 + Ø.3)*RATE) + 1;
            WHILE I < TURNOFF DO
                BEGIN
                FLASH [I] := 1;
                I := I + 1;
                END;
            TURNON := ROUND( J * RATE ) + 1;
            WHILE I < TURNON DO
                BEGIN
                FLASH [I] := Ø;
                I := I + 1;
                END;
            END;
        PERLEN := ROUND(1Ø * RATE);
        END;
MOA   : BEGIN
        TURNOFF := ROUND (Ø.4 * RATE) + 1;
        WHILE I < TURNOFF DO
            BEGIN
            FLASH [I] := 1;
            I := I + 1;
            END;
        TURNON := ROUND (RATE) + 1;
        WHILE I < TURNON DO
            BEGIN
            FLASH [I] := Ø;
            I := I + 1;
```

```
                END;
            TURNOFF := ROUND (3 * RATE) + 1;
            WHILE I < TURNOFF DO
                BEGIN
                FLASH [I] := 1;
                I := I + 1;
                END;
            PERLEN := ROUND(8 * RATE);
            END;
    EINT6 : BEGIN
            TURNOFF := ROUND ( 3.0 * RATE ) + 1;
            WHILE I < TURNOFF DO
                BEGIN
                FLASH [I] := 1;
                I := I + 1;
                END;
            PERLEN := ROUND(6.0 * RATE);
            END;
    OCC4  : BEGIN
            TURNOFF := ROUND ( 3.0 * RATE ) + 1;
            WHILE I < TURNOFF DO
                BEGIN
                FLASH [I] := 1;
                I := I + 1;
                END;
            PERLEN := ROUND(4.0 * RATE);
            END;
    END; {CASE}
WHILE I <= 100 * RATEMULTIPLIER DO
    BEGIN
    FLASH [I] := 0;
    I := I + 1;
    END;
END;
{***************************************************************************}
{***************************************************************************}
{***************************************************************************}
PROCEDURE GET_FLASHED_INTENSITIES (VAR WINDOW     : DATAWINDOWS;
                                   VAR FLASH      : FLASHARRAY;
                                       PERLEN     : INTEGER;
                                       K          : INTEGER);
{Converts fixed on intensity vs time to flashed intensity vs time in WINDOW.
PERLEN is number of data points per flash period in
WINDOW array. Beginning of flash period is indicated by K upon calling this
procedure.  Small number added to intensity assures nonzero during
flash pulse but does not noticably change effective intensity.  This assists
locating pulses in window to determine their effective intensities.}
VAR
    I : INTEGER;
BEGIN
WINDOW.INTENSITY [0] := 0;
I := 1;
WITH WINDOW DO
  WHILE I <= WINDOWLENGTH DO
      BEGIN
      INTENSITY [I] := (INTENSITY [I] + 0.0000001) * FLASH [K];
      I := I + 1;
      K := (K + 1) MOD PERLEN;
      END;
END;
{***************************************************************************}
{***************************************************************************}
{***************************************************************************}
PROCEDURE GET_EVENT_EFI (VAR WINDOW    : DATAWINDOWS;
                             FLASHLEN  : INTEGER);
{Determines maximum effective intensity for all observation events.
Stores value in WINDOW.MAXEFI [event], the EFI for detecting that event.
Also determines minimum EFI of all flash pulses in this event and
stores value in WINDOW.MINEFI [event].  This is the maximum EFI this event
produces such that two complete flash periods are recognized, ie. no missing
pulses in flash period. Uses Schmidt-Clausen method to calculate effective
intensity (calls EFFECTIVE_INTENSITY). MINEFI register in WINDOW must be
reinitialized because flash durations longer than OBSTIMEINC will not
record an efi in first event.}
VAR
    EVENT, EVENTINC, POINTS, I, K : INTEGER;
    EFI    : REAL;     {equivalent fixed intensity, effective intensity}
    TIMEINC : REAL;    {number of seconds between data points in INTEN}
{***************************************************************************}
FUNCTION EFFECTIVE_INTENSITY (VAR WINDOW  : DATAWINDOWS;
                              VAR I       : INTEGER;
                                  TIMEINC : REAL) : REAL;
{INTEN [I] is first nonzero intensity point in flash, INTEN [I-1] is zero.
Integrates until INTEN [I] is zero again and STOP is reached. This function
should not be entered unless at least one data point is guaranteed to be
nonzero, otherwise a divide zero error will occur. Uses trapezoidal formula
to estimate definite integral.}
VAR
    AREA : REAL;    {integral of intensity}
    PEAK : REAL;    {peak intensity value in flash}
BEGIN
AREA := 0;
PEAK := 0;
WITH WINDOW DO
  WHILE INTENSITY [I] > 0 DO
      BEGIN
      AREA := AREA + INTENSITY [I];
      IF INTENSITY [I] > PEAK THEN
          PEAK := INTENSITY [I];
      I := I + 1;
      END;
AREA := AREA * TIMEINC;
EFFECTIVE_INTENSITY := AREA / (0.2 + AREA/PEAK);
END;
{***************************************************************************}
BEGIN
WITH WINDOW DO
  BEGIN
  TIMEINC := 1/SAMPLERATE;               {time between sample points}
  POINTS := ROUND( OBSTIMEINC * SAMPLERATE );  {points per event time increm.}
  FOR EVENT := 1 TO NUMTIMES DO
      BEGIN                        {initialize event efi registers}
      MINEFI [EVENT] := 10000000;
      MAXEFI [EVENT] := 0;
      END;
  EVENT := 1;
  I := 0;
  WHILE EVENT < NUMTIMES DO       {analyze all pulses in flash characteristic}
      BEGIN
      WHILE INTENSITY [I] = 0 DO
          I := I + 1;    {move to first non zero position}
      EFI := EFFECTIVE_INTENSITY (WINDOW, I, TIMEINC);
      EVENT := (I DIV POINTS) + 1;    {determine event cell number}
      IF EVENT <= NUMTIMES THEN
          BEGIN
          IF EFI > MAXEFI [EVENT] THEN
              MAXEFI [EVENT] := EFI;    {for detecting a flash in increment}
          IF EFI < MINEFI [EVENT] THEN
              MINEFI [EVENT] := EFI;   {detecting all flashes}
          END;
      END;  {WHILE EVENT}

              {determine detection efi vs observation time}
  FOR EVENT := 2 TO NUMTIMES DO      {slide maximum up to right}
      IF MAXEFI [EVENT-1] > MAXEFI [EVENT] THEN
          MAXEFI [EVENT] := MAXEFI [EVENT-1];

              {determine recognition efi vs observation time}
  EVENTINC := (2 * FLASHLEN) DIV POINTS;  {difference: end - start}
  IF EVENTINC > 0 THEN   {slide two period window down to left}
      BEGIN
      FOR EVENT := NUMTIMES DOWNTO 1 DO
          BEGIN
          I := EVENT - EVENTINC + 1;
          IF I > 0 THEN
              BEGIN           {get min efi in this 2 period window}
              FOR K := I TO EVENT DO
                  IF MINEFI [K] < MINEFI [EVENT] THEN
                      MINEFI [EVENT] := MINEFI [K];
              END
          ELSE
              MINEFI [EVENT] := 0;
          END;
      END;

  FOR EVENT := 2 TO NUMTIMES DO       {slide maximum up to right}
      IF MINEFI [EVENT-1] > MINEFI [EVENT] THEN
          MINEFI [EVENT] := MINEFI [EVENT-1];

  END; {WITH WINDOW}
END;
{**************************************************************************}
{**************************************************************************}
{**************************************************************************}
PROCEDURE INCREMENT_PROBS (VAR PROB : PROBRECORD;
                           VAR WIN  : DATAWINDOWS);
{Increments the POD and POR arrays in PROB depending on the effective
intensity values in WIN.MAXEFI and WIN.MINEFI arrays. POD is the probability
of detecting at least part of a flash cycle given mariner at specified
distance looks in direction of buoy for a specified length of time.  POR
is defined as the probability a mariner sees two complete and consecutive
flash cycles.  The minimum EFI in every pair of adjacent periods was found in
GET_EVENT_EFI.  The minimum gives distance that both those periods are seen.
The maximum of these minima was then selected.  This gives the distance that
at least one pair of periods in OBSTIME time increments will be seen.  It was
assumed that if one can detect/recognize a flash at distance X in T time, then
one can also detect/recognize same flash in time greater than T.  Same
flash will also be detected/recognized at shorter distances, but may require
less flash periods at these distances.}
VAR
    PREVIOUS : REAL;        {efi in previous period, used for recognition}
    EFIDETEC : REAL;        {efi for detection probability}
    EFIRECOG : REAL;        {efi for recognition probability}
    EFITEST  : REAL;        {minimum efi of current two periods}
    DIST     : INTEGER;     {distance number}
    OBS      : INTEGER;     {observation time numbers}
    P        : INTEGER;     {period counter}
BEGIN
PREVIOUS := WIN.MINEFI [1];
EFIDETEC := 0;
FOR OBS := 1 TO NUMTIMES DO
    BEGIN
    DIST := 1;
    WHILE (WIN.MAXEFI [OBS] >= PROB.EFI[DIST]) AND (DIST <= NUMDISTS) DO
        BEGIN
        PROB.POD [DIST, OBS] := 1.0 + PROB.POD [DIST, OBS];
        DIST := DIST + 1;
        END;
    DIST := 1;
    WHILE (WIN.MINEFI [OBS] >= PROB.EFI[DIST]) AND (DIST <= NUMDISTS) DO
        BEGIN
        PROB.POR [DIST, OBS] := 1.0 + PROB.POR [DIST, OBS];
        DIST := DIST + 1;
        END;
    END;
END;
{*****************************************************************************}
{*****************************************************************************}
{*****************************************************************************}
PROCEDURE NORMALIZE_PROBS (VAR PROB : PROBRECORD;
                               NUMWINDOWS : INTEGER);
{normalizes all the probabilities by the number of windows that were examined
in the data record}
VAR
    DIST, T : INTEGER;
BEGIN
WITH PROB DO
    FOR DIST:=1 TO NUMDISTS DO
        BEGIN
        FOR T:=1 TO NUMTIMES DO
            BEGIN
            POD [DIST, T] := POD [DIST, T] * 100 / NUMWINDOWS;
            POR [DIST, T] := POR [DIST, T] * 100 / NUMWINDOWS;
            END;
        END;
END;
{****************************************************************************}
{****************************************************************************}
{****************************************************************************}
PROCEDURE PRINT_PROBABILITIES (VAR OUTDISK : PROBFILE;
                                   PROB : PROBRECORD;
                                   HEADER : HEADRECORD;
                                   FLASHCHAR : CHARACTERISTICS);
{Writes Probabilities to floppy disk and printer. If first Prob record then
procedure determines disk file name from header data and opens disk file for
writing.  If last Prob record it closes disk. Roll period is printed for
simulated data.  Simulated vs actual is determined by whether or not
rollamplitude equals zero.}
VAR
    P : 1..2;
    I, D, T : INTEGER;
    FILENAME     : STRING[20];
    S : STRING [80];
BEGIN
IF FLASHCHAR = FL6 THEN   {get file name and open disk file}
    BEGIN
    FILENAME := '#4:D';
    STRWRITE (FILENAME,5,T,HEADER.MONTH:0);
    FILENAME := FILENAME + '_';
    STRWRITE (FILENAME,T+1,T,HEADER.DAY:0);
    FILENAME := FILENAME + 'H';
    STRWRITE (FILENAME,T+1,T,HEADER.STARTTIME DIV 360000 :0);
    REWRITE (OUTDISK, FILENAME);
    END;
WRITE (OUTDISK, PROB);
IF FLASHCHAR = OCC4 THEN     {close if last prob record}
    CLOSE (OUTDISK, 'SAVE');
FOR P:=1 TO 2 DO
    BEGIN
    WRITELN(PRIN, ESC,'&k3S');    {expanded-compressed 10.7 cpi}
    WRITELN(PRIN);
    WRITELN(PRIN);
    IF P=1 THEN
        WRITELN(PRIN,' ':22,'Probability of Detection')
    ELSE
        WRITELN(PRIN,' ':21,'Probability of Recognition');
    CASE FLASHCHAR OF
        FL6    : WRITELN (PRIN,' ':29,'FL. 6 (0.6)');
        FL4    : WRITELN (PRIN,' ':29,'FL. 4 (0.4)');
        FL25   : WRITELN (PRIN,' ':28,'FL. 2.5 (0.3)');
        QKFL   : WRITELN (PRIN,' ':31,'QK. FL.');
        GPFL5  : WRITELN (PRIN,' ':30,'GP. FL. 5');
        GPFL6  : WRITELN (PRIN,' ':30,'GP. FL. 6');
        IQKFL  : WRITELN (PRIN,' ':30,'I. QK. FL.');
        MOA    : WRITELN (PRIN,' ':31,'MO. (A)');
        EINT6  : WRITELN (PRIN,' ':30,'E. Int. 6');
        OCC4   : WRITELN (PRIN,' ':32,'Occ. 4');
        END; {CASE}
    WRITELN (PRIN);
    WRITELN (PRIN);
WRITELN (PRIN);
WRITELN (PRIN, ESC,'&k0S');     {normal print 12cpı}
WRITELN (PRIN,'Distance',' ':25,'Observation Time (s)');
WRITE (PRIN,' ':1,'Nt.Mi.      ');
WRITE (PRIN, ESC,'&k2S');       {compressed print 21.3 cpi}
FOR I:=1 TO NUMTIMES DO WRITE (PRIN,(I*OBSTIMEINC):5,' ':3);
WRITELN (PRIN);
WITH PROB DO
  FOR D:=1 TO NUMDISTS DO
    BEGIN
    WRITE (PRIN, ESC,'&k0S');   {normal print}
    WRITE (PRIN,' ':2,(D * DISTANCEINC):3:1,' ':3);
    WRITE (PRIN, ESC,'&k2S');   {compressed}
    WRITE (PRIN,' ':6);
    IF P=1 THEN
        BEGIN
        FOR I:=1 TO NUMTIMES DO WRITE (PRIN,POD [D,I] :7:0,'%')
        END
    ELSE
        BEGIN
        FOR I:=1 TO NUMTIMES DO WRITE (PRIN,POR [D,I] :7:0,'%')
        END;
    WRITELN(PRIN);
    WRITELN(PRIN);
    END;
WRITE (PRIN, ESC,'&k0S');  {normal print}
FOR I:=1 TO 4 DO WRITELN (PRIN);
WITH HEADER DO
    BEGIN
    S := '';
    FOR I:=1 TO 80 DO STRWRITE (S,I,T,' ');
    STRWRITE (S,5,T,'Date: ');
    STRWRITE (S,11,T,MONTH:2);
    STRWRITE (S,13,T,'/');
    STRWRITE (S,14,T,DAY:2);
    STRWRITE (S,16,T,'/');
    STRWRITE (S,17,T,YEAR:2);
    STRWRITE (S,45,T,'Time: ');
    STRWRITE (S,51,T,STARTTIME DIV 360000 :2);
    STRWRITE (S,53,T,':');
    STRWRITE (S,54,T,(STARTTIME MOD 360000) DIV 6000 :2);
    STRWRITE (S,56,T,':');
    STRWRITE (S,57,T,((STARTTIME MOD 360000) MOD 6000) DIV 100 :2);
    WRITELN (PRIN, S);
    FOR I:=1 TO 80 DO STRWRITE (S,I,T,' ');
    STRWRITE (S,5,T,'Sample Rate: ');
    STRWRITE (S,18,T,SAMPLERATE:2:2);
    STRWRITE (S,21,T,'Hz');
    STRWRITE (S,45,T,'Lantern File: ');
    STRWRITE (S,59,T,LANTERNFILE);
    WRITELN (PRIN, S);
    FOR I:=1 TO 80 DO STRWRITE (S,I,T,' ');
    IF ROLLPERIOD<>0 THEN       {data is simulated}
        BEGIN
        STRWRITE (S,5,T,'Roll Amplitude: ');
        STRWRITE (S,21,T,ROLLAMPLITUDE:2:0);
```

```
            STRWRITE (S,24,T,' deg.');
            STRWRITE (S,45,T,'Sinusoidal Roll Period: ');
            STRWRITE (S,69,T,ROLLPERIOD:2:1);
            STRWRITE (S,73,T,' sec');
            END
        ELSE
            BEGIN
            STRWRITE (S,5,T,'RMS Roll Amplitude: ');
            STRWRITE (S,25,T,ROLLAMPLITUDE:2:1);
            STRWRITE (S,29,T,' deg.');
            STRWRITE (S,45,T,'Mean roll angle: ');
            STRWRITE (S,62,T,MEANROLLANGLE:2:1);
            STRWRITE (S,67,T,'deg.');
            END;
        WRITELN (PRIN, S);
        END;
    IF FLASHCHAR = FL6 THEN
        BEGIN
        FOR I:=1 TO 80 DO STRWRITE (S,I,T,' ');
        STRWRITE (S,5,T,'Output Data File: ');
        STRWRITE (S,23,T,FILENAME);
        WRITELN (PRIN, S);
        END;
    PAGE (PRIN);
    END;
END;
{************************************************************************}
{************************************************************************}
{************************************************************************}
BEGIN  {BUOY_PROBABILITIES}
REWRITE (PRIN, 'PRINTER:');
INPUT_USER_DATA (FILENAMES, LANTERNFILE);
I := 1;    {counter for data blocks on disk}
WHILE FILENAMES [I] <> 'NONE' DO   {repeat for all data sets}
    BEGIN
    STOREDATA_IN_RAMFILE (FILENAMES [I], LANTERNFILE, HEADER, NUMREADINGS);
    RANGE := (NUMREADINGS - WINDOWLENGTH) DIV 2;    {for random pos}
    WINDOW.SAMPLERATE := HEADER.SAMPLERATE * RATEMULTIPLIER;
    FOR FLASHCHAR := FL6 TO OCC4 DO    {repeat for all characteristics}
        BEGIN
        SEED := 1;    {seed for random number generator}
        INITPROB_RECORD (PROBABILITY);   {zero prob array}
        FLASH_PROPERTIES (FLASH, FLASHLEN, FLASHCHAR, WINDOW.SAMPLERATE);
        FOR W := 1 TO NUMWINDOWS DO
            BEGIN
            FILEPOS := 2 * RAND (SEED, RANGE) + 1;
            GET_DATA_WINDOW (WINDOW, FILEPOS);
            WITH WINDOW DO    {zero maxefi and minefi arrays}
                FOR TIME := 1 TO NUMTIMES DO
                    BEGIN
                    MAXEFI [TIME] := 0;
                    MINEFI [TIME] := 0;
                    END;
            STARTPOS := RAND (SEED, FLASHLEN);
            GET_FLASHED_INTENSITIES (WINDOW, FLASH, FLASHLEN, STARTPOS);
            GET_EVENT_EFI (WINDOW, FLASHLEN);
            {PLOT_WINDOW (WINDOW);}     {for testing}

            INCREMENT_PROBS (PROBABILITY, WINDOW);
            END;
        NORMALIZE_PROBS (PROBABILITY, NUMWINDOWS);
        PRINT_PROBABILITIES (OUTDISK, PROBABILITY, HEADER, FLASHCHAR);
        END; {for flashchar}
    I := I + 1;
    END; {while datalocation}
CLOSE (PRIN);
END.
```

Program LanternProfileMaker

```
PROGRAM LanternProfileMaker (INPUT, OUTPUT);
{Generates lantern beam profiles with a Gaussian shape, a user specified
vertical divergence (FWHM), with the same total flux as an existing
lanternfile used in PROGRAM BUOY_PROBABILITIES.  by Dan Brown, 4/2/87}
CONST
    lanternfile = '#3:LANT.REAL';
    DegToRad = 0.017453293;
TYPE
    filenameType = STRING [20];
VAR
    P : TEXT;
    filename : filenameType;          {name of lantern file to integrate}
    flux : REAL;                      {total flux in integrated profile}
    fullWidth : REAL;                     {full width half max in degrees}
    limitLevel : REAL;                {fraction of peak at limits of integ}
    k : REAL;                         {constant for gaussian exponential}
    integral : REAL;                  {integral of gaussian}
    peak : REAL;                      {peak of gaussian}
    s : STRING [10];
    i : INTEGER;
{**************************************************************************}
{**************************************************************************}
PROCEDURE IntegrateLanternFile (filename : filenameType;
                                limLevel : REAL;
                                VAR flux : REAL);
TYPE
    lantRecord = RECORD
                   angle     : REAL;
                   intensity : REAL;
                 END;
    lantArray = ARRAY [0..455] OF lantRecord;

VAR
    lanternData : lantArray;
    maxIntensity : REAL;
    angleInc : REAL;
    f : FILE OF REAL;
    i : INTEGER;
{**************************************************************************}
PROCEDURE GetLanternData (VAR lanternData : lantArray;
                          VAR maxInten : REAL);
{reads lantern file on disk}
VAR
    i : integer;
BEGIN
lanternData [0].angle := -95.0;
lanternData [0].intensity := 0.0;
maxInten := 0.0;
i := 1;
WHILE (NOT EOF(f)) AND (i < 455) DO
    WITH lanternData [i] DO
        BEGIN
        READ (f, angle, intensity);
        IF intensity > maxInten THEN
            maxInten := intensity;
        i := i + 1;
        END;
WHILE i <= 455 DO
    BEGIN
    lanternData [i].angle := 95;
    lanternData [i].intensity := 0.0;
    i := i + 1;
    END;
END;
{***************************************************************************}
PROCEDURE PrintLanternData (VAR lanternData : lantArray);
VAR
    p : TEXT;
    i : INTEGER;
BEGIN
i := 0;
REWRITE (p, 'PRINTER:');
WHILE lanternData [i].angle < 90 DO
    BEGIN
    i := i + 1;
    WRITELN (p, lanternData [i].angle :5:1, lanternData [i].intensity:10:2);
    END;
END;
{***************************************************************************}
BEGIN {IntegrateLanternFile}
RESET (f, filename);
GetLanternData (lanternData, maxIntensity);
CLOSE (f);
i := 1;
WHILE lanternData [i].angle < 0 DO i := i + 1;
limLevel := limLevel * maxIntensity;
WHILE lanternData [i].intensity > limLevel DO i := i - 1;
angleInc := lanternData [i+1].angle - lanternData [i].angle;  {in deg.}
WITH lanternData [i] DO
    flux := intensity * cos (angle * DegToRad);
i := i + 1;
WITH lanternData [i] DO
    flux := flux + 4 * intensity * cos (angle * DegToRad);
i := i + 1;
WHILE lanternData[i].intensity > limLevel DO
    BEGIN
    WITH lanternData [i] DO
        flux := flux + 2 * intensity * cos (angle * DegToRad);
    i := i + 1;
    WITH lanternData [i] DO
        flux := flux + 4 * intensity * cos (angle * DegToRad);
    i := i + 1;
    END;
WITH lanternData [i] DO
    flux := flux + intensity * cos (angle * DegToRad);
flux := angleInc * DegToRad * flux / 3;
END;
{***************************************************************************}
{***************************************************************************}
FUNCTION Gauss ( k, x : REAL) : REAL;
BEGIN
Gauss := exp ( k * x * x );
```

```
END;
{****************************************************************************}
{****************************************************************************}
PROCEDURE IntegrateGaussian (hwhm,   {half width half max}
                             lim : REAL;      {% peak at limit of integrate}
                             VAR k : REAL;        {const for exponential}
                             VAR sum : REAL);
{finds total flux in a gaussian beam profile}
CONST
    nMax = 400;
VAR
    n : INTEGER;
    middle : INTEGER;
    angle : REAL;
    angleLimit : REAL;                {limits of integration of gaussian}
    angleInc : REAL;
BEGIN {IntegrateGaussian}
k :=  LN (0.5) / hwhm / hwhm ;
angleLimit := SQRT( LN( lim ) / k );
angleInc := 2 * angleLimit / nMax;
angle := -angleLimit;
sum := Gauss(k, angle) * cos (angle * DegToRad);
angle := angle + angleInc;
sum := sum + 4 * Gauss(k, angle) * cos (angle * DegToRad);
middle := nMax DIV 2;
n := 2;
WHILE n < nMax DO
    BEGIN
    angle := (n - middle) * angleInc;
    sum := sum + 2 * Gauss( k, angle ) * cos (angle * DegToRad);
    n := n + 1;
    angle := (n - middle) * angleInc;
    sum := sum + 4 * Gauss( k, angle ) * cos (angle * DegToRad);
    n := n + 1;
    END;
angle := (n - middle) * angleInc;
sum := sum + Gauss( k, angle ) * cos (angle * DegToRad);
sum := sum * angleInc * DegToRad / 3;
END;
{****************************************************************************}
{****************************************************************************}
PROCEDURE StoreGaussProfile (k, peak, fullWidth : REAL);
CONST
    angleInc = 0.2;       {degrees}
VAR
    filename : filenameType;
    f : FILE OF REAL;
    angle : REAL;
    i : INTEGER;
BEGIN
filename := '#3:GAU';
STRWRITE (filename, 7, i, ROUND(fullWidth):0 );
filename := filename + '.REAL';
WRITELN ('File name is: ',filename);
REWRITE (f, filename);
i := ROUND (-45 / angleInc);
```

```
WHILE angle < 45.0 DO
    BEGIN
    angle := i * angleInc;
    WRITE (f, angle, peak * Gauss(k,angle) );
    i := i + 1;
    END;
CLOSE (f, 'SAVE');
END;
{**************************************************************************}
{**************************************************************************}
BEGIN  {MAIN}
REWRITE (P, 'PRINTER:');
WRITELN ('Enter file name (including volume name) of lanternfile');
WRITELN ('to match output flux to.');
READLN (filename);
WRITELN ('Enter fraction of peak intensity at limits of integration');
WRITELN ('default is 0.01');
limitLevel := 0.01;
s := '';
READLN (s);
IF s <> '' THEN
    STRREAD (s,1,i,limitLevel);
WRITELN (limitLevel);
WRITELN ('Enter full vertical width (in degrees) of generated profile');
WRITELN ('at 50% intensity');
READLN (fullWidth);
IntegrateLanternFile (filename, limitLevel, flux);
WRITELN ('flux = ',flux);
IntegrateGaussian (fullWidth / 2, limitLevel, k, integral);
peak := flux / integral;
WRITELN ('PEAK INTENSITY OF GAUSSIAN = ',peak);
WRITELN ('k = ',k);
StoreGaussProfile (k, peak, fullWidth);
END.
```

Program **FastFourierTransform**

```
PROGRAM FastFourierTransform(INPUT,OUTPUT);
{Performs a Fast Fourier Transform on any number of data points equal RADIX
to power of GAMMA. User inputs approx. number of data points.  Program
calculates global variables GAMMA and N (number of data points). User may
convolve data with a five number kernel prior to FFT to remove noise,
differentiate, etc.  Five numbers (or weights) are user input.  Convolution
is a running average of current point and two points on either side of current
with weighting defined in kernel.  Windowing of time domain is also allowed
after done convolving.  Four windows are provided: rectangular, Hamming,
Hanning, and Hanning Squared.  User may also take average of several FFT's by
specifying that more than one data files are to be transformed.
BY DAN BROWN, April 1987}
IMPORT
    IODECLARATIONS,
    SYSDEVS,
    DGL_LIB,
    DGL_INQ;
CONST
    RADIX = 2;  {base or radix of FFT routine}
    NMAX = 16384;
    PI = 3.141592654;
TYPE
    COMPLEX_NUMBERS = RECORD
                        RE : REAL;
                        IM : REAL;
                      END;
    DATA_ARRAY = ARRAY[0..NMAX] OF COMPLEX_NUMBERS;
    POINTER = ^DATA_ARRAY;
    FILESTRING = STRING[20];
    PLOT_TYPES = ( MAGNITUDE, PHASE, TIMESERIES );
    messageType = STRING [60];
VAR
    GAMMA : INTEGER;
    N : INTEGER;  {number of data points is RADIX to the power of GAMMA}
    X : POINTER;                {points to data array}
    HEAP : POINTER;             {used to restore heap memory}
    PTR : TEXT;                 {file name for printer}
    numfiles : INTEGER;
    FILENAME : FILESTRING;      {file name for data file on disc}
    TIMEINC : REAL;             {time in seconds between data samples}
    VARIANCE : REAL;            {mean square amplitude}
    MAXFREQ : REAL;
    IMAX : INTEGER;
    i : INTEGER;

{*************************************************************************}
{*************************************************************************}
PROCEDURE CLEARSCREEN;
{clears CRT of all text and spaces cursor}
BEGIN
WRITELN(CHR(12));   {homes cursor and clears screen}
WRITELN;
WRITE ('        ');
END;
{************************************************************************
```

AD-A194 665   PROBABILITIES OF DETECTION AND RECOGNITION OF FLASHING   2/2
LIGHTS ON ROLLING BUOYS(U) COAST GUARD RESEARCH AND
DEVELOPMENT CENTER GROTON CT D M BROWN AUG 87
UNCLASSIFIED   CGR/DC-06/87 USCG-D-10-88   F/G 17/7.2   NL

1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.2
1.4
1.6

```
{**************************************************************************}
FUNCTION AnswerYes (message : messageType) : BOOLEAN;
{returns TRUE if anwer with a Y (yes), returns FALSE if with N (no)}
VAR
    ans : CHAR;
    ansValid : BOOLEAN;
BEGIN
CLEARSCREEN;
ansValid := FALSE;
REPEAT
    WRITE (message);
    WRITE (' (Y or N)  ');
    READ (ans);
    CLEARSCREEN;
    CASE ans OF
      'Y','y','N','n' : ansValid := TRUE;
    OTHERWISE  END; {CASE}
UNTIL ansValid;
CASE ans OF
  'Y','y' : AnswerYes := TRUE;
  'N','n' : AnswerYes := FALSE;
  OTHERWISE
    END;
END;
{**************************************************************************}
{**************************************************************************}
PROCEDURE WriteMessage (message : messageType);
VAR
    i : INTEGER;
BEGIN
CLEARSCREEN;
FOR i := 1 TO 10 DO WRITELN ('          ');
WRITELN (message);
END;
{**************************************************************************}
{**************************************************************************}
FUNCTION DIGIT_REVERSE(Z, GAMMA : INTEGER) : INTEGER;
{forms the bit reversed number of Z.  Example:  0010 -> 0100.}
VAR
    I, Y : INTEGER;
BEGIN
Y:=0;
FOR I:=1 TO GAMMA DO           {gamma is also number of digits}
    BEGIN
    Y := RADIX * Y + (Z MOD RADIX);   {shift left, add rightmost digit}
    Z := Z DIV RADIX;                 {shift right}
    END;
DIGIT_REVERSE:=Y;
END;
{**************************************************************************}
{**************************************************************************}
PROCEDURE LOAD_DATA_ARRAY( VAR X : POINTER;
                               N : INTEGER;
                           VAR TIMEINC : REAL;
                           VAR VARIANCE : REAL;
                               FILENAME : FILESTRING);
```

```
{reads in data from a disc file and stores data into array X.  Procedure
assumes all data in file is real, no imaginary part.  Thus zeros are put
in imaginary part of array.  Data file can have more or less than N points.}
VAR
    S : STRING[80];           {data string from ASCII file}
    F : TEXT;
    I, T : INTEGER;
    NUMLINES : INTEGER;
    MEAN : REAL;
BEGIN
RESET(F,FILENAME);
WRITE ('number of lines of data file to skip?  ');
READLN (NUMLINES);
FOR I:=1 TO NUMLINES DO
    BEGIN
    READLN (F, S);
    WRITELN (S);
    END;
WRITELN;
WRITELN;
WRITE ('sampling rate of data (Hz)?  ');
READLN (TIMEINC);
TIMEINC := 1/TIMEINC;       {no. seconds between points}
MEAN := 0;
VARIANCE := 0;
FOR I:=0 TO N-1 DO
    BEGIN
    IF NOT EOF(F) THEN
        BEGIN
        READLN(F,X^[I].RE);
        VARIANCE := VARIANCE + X^[I].RE * X^[I].RE;
        MEAN := MEAN + X^[I].RE;
        END
    ELSE
        X^[I].RE := 0.0;      {for files with less than N points}

    X^[I].IM := 0.0;
    END;
CLOSE(F);
CLEARSCREEN;
WRITELN (PTR, 'TOTAL ENERGY IN TIME DOMAIN = ',VARIANCE);
VARIANCE := VARIANCE / N;
WRITELN (PTR, 'VARIANCE OF WAVE AMPLITUDE = ',VARIANCE);
WRITELN (PTR, 'MEAN OF WAVE AMPLITUDE = ',MEAN / N);
END;
{**************************************************************************}
{**************************************************************************}
PROCEDURE CONVOLVE (VAR X : POINTER; N : INTEGER);
{Convolves data with a five number kernel}
VAR
    kernel : ARRAY [-2..+2] OF REAL;
    buffer : ARRAY [-2..0] OF REAL;
    i, j, k : INTEGER;
    sum : REAL;
    F : TEXT;
BEGIN
```

```
WRITELN ('input convolution kernel');
FOR i:= -2 TO 2 DO
    BEGIN
    WRITE (i,' ->');
    READLN (kernel [i]);
    END;
WRITELN (PTR,'CONVOLUTION KERNEL:');
FOR i:= -2 TO 2 DO WRITELN (PTR, kernel [i]);
FOR k := -2 to 0 DO
    buffer [k] := 0.0;              {init buffer}
WriteMessage ('WAIT ! CONVOLVING DATA');
FOR i:= 0 TO N-3 DO
    BEGIN
    sum := 0;
    j := 3;
    WHILE (j > -2) AND (i+j > 0) DO
        BEGIN
        j := j - 1;
        sum := sum + kernel [j] * X^[i+j].RE;
        END;
    IF i-2 > 0 THEN
        X^[i-3].RE := buffer [-2];      {remove last entry from buffer}
    FOR k:= -2 TO -1 DO
        buffer [k] := buffer [k+1];     {slide entries down one}
    buffer [0] := sum;
    END;
FOR k:=-2 TO 0 DO
    X^[N-3+k].RE := buffer [k];             {empty buffer}
CLEARSCREEN;

{REWRITE (F, '#3:CONVOL.TEXT');
FOR k:=0 TO N-1 DO
    WRITELN (F, X^[k].RE);
CLOSE (F, 'SAVE');}

END;
{****************************************************************************}
{****************************************************************************}
FUNCTION SignificantDigits (num :REAL; digits : INTEGER) : REAL;
{returns "num" rounded to "digits" significant digits}
VAR
    comp : INTEGER;
BEGIN
comp := 10;       {integer with "digits" significant digits}
WHILE digits > 1 DO
    BEGIN
    comp := comp * 10;
    digits := digits - 1;
    END;
digits := 0;     {number of digits to shift num after rounding}
WHILE num < comp DO
    BEGIN
    num := num * 10;
    digits := digits - 1;
    END;
WHILE num > comp DO
```

```
VAR
    CHARWIDTH, CHARHEIGHT : REAL;
    TITLE                 : STRING[80];
    X, DX, Y, DY          : REAL;
    XLABEL, YLABEL        : REAL;
    I, J, K               : INTEGER;
BEGIN
CHARWIDTH := (XMAX - XMIN) / 50;
CHARHEIGHT := (YMAX - YMIN) / 25;
SET_CHAR_SIZE(CHARWIDTH,CHARHEIGHT);     {sets constant relative size}
CASE PLT OF
  TIMESERIES,
  MAGNITUDE   : BEGIN
               SET_TEXT_ROT (0,1);
               MOVE(XMAX,0);
               LINE(0,0);
               X := 0;
               DX := (XMAX - XMIN ) / 40;            {Label x axis}
               FOR I:=1 TO 4 DO      {major tick marks, horizontal axis}
                   BEGIN
                   FOR J:=1 TO 10 DO    {minor tick marks}
                       BEGIN
                       X := X + DX;
                       MOVE(X,0);
                       LINE(X,-0.5*CHARHEIGHT);
                       END;
                   LINE(X,-CHARHEIGHT);
                   TITLE:='';                     {label major ticks}
                   IF PLT = TIMESERIES THEN
                       BEGIN
                       XLABEL := X * TIMEINC;             {time}
                       XLABEL := SignificantDigits (XLABEL, 3);
                       STRWRITE(TITLE,1,K,XLABEL:3:2);
                       END
                   ELSE
                       BEGIN
                       XLABEL := X / TIMEINC / N;     {frequency}
                       XLABEL := SignificantDigits (XLABEL, 3);
                       STRWRITE(TITLE,1,K,XLABEL:4:2);
                       END;
                   MOVE(X,CHARHEIGHT);
                   GTEXT(TITLE);
                   END;
               SET_TEXT_ROT (1,0);
               MOVE(0,YMAX);
               LINE(0,YMIN);
               LINE(-CHARWIDTH,YMIN);
               Y := YMIN;
               DY := (YMAX - YMIN)/10;        {vertical axis tic marks}
               FOR I:=1 TO 10 DO
                   BEGIN
                   Y := Y + DY;
                   MOVE(0,Y);
                   LINE(-CHARWIDTH,Y);
                   END;
               TITLE:='';
               YLABEL := SignificantDigits (YMAX, 2);
               STRWRITE(TITLE,1,K,YMAX:4:2);
               MOVE(-STRLEN(TITLE)*CHARWIDTH,YMAX);
               GTEXT(TITLE);
               END;
  PHASE      : BEGIN
               SET_TEXT_ROT (0,1);
               MOVE(XMAX,0);
               LINE(XMIN,0);
               MOVE(0,YMIN);
               LINE(0,YMAX);
               X := XMIN;
               DX := (XMAX - XMIN ) / 80;             {Label x axis}
               FOR I:=1 TO 8 DO      {major tick marks}
                   BEGIN
                   FOR J:=1 TO 10 DO    {minor tick marks}
                       BEGIN
                       X := X + DX;
                       MOVE(X,0);
                       LINE(X,-0.5*CHARHEIGHT);
                       END;
                   LINE(X,-CHARHEIGHT);
                   TITLE:='';                  {label major ticks}
                   XLABEL := X / XMAX / TIMEINC / 2;     {frequency}
                   STRWRITE(TITLE,1,K,XLABEL:0:2);
                   MOVE(X,CHARHEIGHT);
                   GTEXT(TITLE)
                   END;
               SET_TEXT_ROT (1,0);
               Y := YMIN;                   {label y axis}
               DY := PI/18;
               MOVE(-CHARWIDTH,YMIN);
               LINE(CHARWIDTH,YMIN);
               TITLE := '-90 deg';
               GTEXT(TITLE);
               FOR I:=1 TO 18 DO
                   BEGIN
                   Y := Y + DY;
                   MOVE(-CHARWIDTH/2,Y);
                   LINE(CHARWIDTH/2,Y);
                   END;
               LINE(CHARWIDTH,YMAX);
               TITLE:='+90 deg';
               GTEXT(TITLE);
               END;
  OTHERWISE
  END; {CASE}
CASE PLT OF
  TIMESERIES : TITLE := 'Time Series';
  MAGNITUDE  : TITLE := 'Magnitude';
  PHASE      : TITLE := 'Phase angle';
  END; {CASE}
MOVE ( (XMAX - XMIN - STRLEN(TITLE)*CHARWIDTH)/3 + XMIN, YMAX );
GTEXT(TITLE);
END;
{*************************************************************************}
```

```
PROCEDURE SET_USER_SCALE (VAR P     : POINTER;
                              PLOT : PLOT_TYPES;
                              XMAX : INTEGER;
                              TIMEINC : REAL;
                          VAR MAXVALUE : REAL);
VAR
    VALUE : REAL;
    I : INTEGER;
    maxPoint : INTEGER;
    XMIN, YMIN, YMAX : REAL;
    ENERGY : REAL;
BEGIN
MAXVALUE := 0;
ENERGY := 0;
FOR I:=0 TO N-1 DO          {find maximum value in data array}
    BEGIN
    VALUE := P^[I].RE*P^[I].RE + P^[I].IM*P^[I].IM;
    ENERGY := ENERGY + VALUE;
    IF VALUE>MAXVALUE THEN
        BEGIN
        MAXVALUE:=VALUE;
        maxPoint := I;
        END;
    END;
MAXVALUE := SQRT(MAXVALUE);
WRITELN (PTR, 'MAX VALUE = ',MAXVALUE,' AT I=',maxPoint);
CASE PLOT OF
  TIMESERIES : BEGIN
               {WRITELN (PTR, 'TOTAL ENERGY IN TIME DOMAIN = ',ENERGY);}
               XMIN:=0.0;
               YMIN:= -MAXVALUE;
               YMAX:= MAXVALUE;
               END;
  MAGNITUDE  : BEGIN
               {WRITELN (PTR,'TOTAL ENERGY IN FREQUENCY DOMAIN = ',
                                     ENERGY / N);}
               XMIN:= 0.0;
               YMIN:=0.0;
               YMAX:=MAXVALUE;
               END;
  PHASE      : BEGIN
               XMIN:=-XMAX;
               YMIN:=-PI/2;
               YMAX:=PI/2;
               END;
  END; {CASE}
SET_WINDOW(XMIN,XMAX,YMIN,YMAX);          {Sets user coord. scale}
LABEL_PLOT (PLOT,XMIN,XMAX,YMIN,YMAX);
END;
{****************************************************************************}
BEGIN   {PLOT_DATA}
INITIALIZE_PLOT;
IF PLOT=TIMESERIES THEN
    BEGIN          {****** TIME SERIES PLOT *****}
    SET_USER_SCALE (P,TIMESERIES,IMAX,TIMEINC,MAXMAG);
    MOVE(0, P^[0].RE);
```

```
        FOR I:=1 TO IMAX DO
            LINE(I,P^[I].RE);
        IF AnswerYes ('HARD COPY OF PLOT?') THEN
            CALL(DUMPGRAPHICSHOOK);
        CLEAR_DISPLAY;
        END
    ELSE
        BEGIN
                  {***** MAGNITUDE PLOT *****}
        SET_USER_SCALE (P,MAGNITUDE,IMAX,TIMEINC,MAXMAG);
        MOVE(0,SQRT( P^[0].RE*P^[0].RE + P^[0].IM*P^[0].IM));
        FOR I:=1 TO IMAX DO
            BEGIN
            Y := SQRT( P^[I].RE*P^[I].RE + P^[I].IM*P^[I].IM );
            LINE(I,Y);
            END;
        IF AnswerYes ('HARD COPY OF PLOT?') THEN
            BEGIN
            CALL(DUMPGRAPHICSHOOK);
            PAGE (PTR);
            END;
        CLEAR_DISPLAY;

                       {***** PHASE PLOT *****}
        SET_USER_SCALE (P,PHASE,IMAX,TIMEINC,MAXMAG);
        HALF_N := N DIV 2;
        I:=N - IMAX;          {max negative freq starts at N/2 or greater}
        FIRSTPOINT := TRUE;
        REPEAT
            X := I - (I DIV HALF_N)*N;   {I-N when I>N/2, I when I<N/2}
            IF P^[I].IM*P^[I].IM + P^[I].RE*P^[I].RE > 0.01*MAXMAG THEN
                BEGIN           {non zero phase}
                IF P^[I].RE=0 THEN     {90 deg phase}
                    BEGIN
                    IF P^[I].IM<0 THEN Y:= -PI/2
                    ELSE Y:= PI/2;
                    END
                ELSE                   { phase < 90 }
                    BEGIN
                    Y := ARCTAN(P^[I].IM/P^[I].RE);
                    END;
                END
            ELSE              {zero phase}
                BEGIN
                Y:=0;
                END;
            IF FIRSTPOINT THEN
                BEGIN
                MOVE(X,Y);
                FIRSTPOINT := FALSE;
                END
            ELSE
                LINE(X,Y);
            I := (I+1) MOD N;   {start midway and wrap around}

        UNTIL I=IMAX;
    IF AnswerYes ('HARD COPY OF PLOT?') THEN
        BEGIN
        CALL(DUMPGRAPHICSHOOK);
        PAGE (PTR);
        END;
    CLEAR_DISPLAY;
    END;
GRAPHICS_TERM;
END;
{**************************************************************************}
{**************************************************************************}
PROCEDURE ORDER_DATA_ARRAY(VAR X : POINTER; GAMMA, N : INTEGER);
{unscrambles the data array}
VAR
    I : INTEGER;  {current node}
    J : INTEGER;  {node to be exchanged}
    NODE : COMPLEX_NUMBERS;
BEGIN
FOR I:=0 TO N-1 DO
    BEGIN
    J := DIGIT_REVERSE(I, GAMMA);
    IF J<I THEN          {exchange only those nodes not yet exchanged}
        BEGIN
        NODE := X^[I];
        X^[I] := X^[J];
        X^[J] := NODE;
        END;
    END;
END;
{**************************************************************************}
{**************************************************************************}
PROCEDURE WINDOW_DATA (VAR X : POINTER; N : INTEGER);
CONST
    RETURN = CHR(32);
    UP = CHR(31);
    DOWN = CHR(10);
    numSelects = 4;
VAR
    C : CHAR;
    I, T : INTEGER;
    ITEM : 0..4;
    SELECTION : (RECTANGLE, HAMMING, HANNING, SQHANNING);
    NUM : REAL;
{**************************************************************************}
PROCEDURE SETCURSOR(J : INTEGER);
{positions cursor next to Jth item in selection}
VAR
    I : INTEGER;
BEGIN
WRITE(CHR(8));       {move cursor left one position if possible}
WRITE(' ');
WRITE(CHR(1));       {home cursor}
FOR I:=1 TO 3 DO WRITE(CHR(10));      {move cursor down}
FOR I:=1 TO 26 DO WRITE(CHR(28));     {move cursor to right}
WHILE J<>0 DO
```

```
    BEGIN          {position cursor next to selection}
    J:=J-1;
    WRITE(CHR(10));     {move cursor down one line}
    END;
WRITE('<');         {write the pointer}
END;
{***************************************************************************}
BEGIN
CLEARSCREEN;
WRITELN( CHR(1) );
WRITELN('Select a data filter window by setting pointer with up or down');
WRITELN('arrow keys.  Then press <ENTER> key to make selection');
WRITELN;
WRITELN('*          Rectangle     *');
WRITELN('*            Hamming     *');
WRITELN('*            Hanning     *');
WRITELN('*    Hanning Squared     *');
ITEM := 1;
WRITELN;
SETCURSOR(ITEM);
REPEAT
    READ(C);
    CASE C OF
      RETURN : CASE ITEM OF
                 1 : SELECTION := RECTANGLE;
                 2 : SELECTION := HAMMING;
                 3 : SELECTION := HANNING;
                 4 : SELECTION := SQHANNING;
                 END;
          UP : BEGIN
               ITEM := ((ITEM-2+numSelects) MOD numSelects)+1;
               WRITE(CHR(10));          {negate key press with down arrow}
               END;
        DOWN : BEGIN
               ITEM := (ITEM MOD numSelects) + 1;
               WRITE(CHR(31));          {negate key press with up arrow}
               END;
       OTHERWISE
       END; {CASE}
    SETCURSOR(ITEM);          {position cursor}
UNTIL C=RETURN;
CLEARSCREEN;
CASE SELECTION OF
  HAMMING   : BEGIN
              WRITELN ('WAIT !  WINDOWING WITH HAMMING');
              FOR T:=0 TO N-1 DO
                  X^[T].RE := X^[T].RE * (0.08+0.46-0.46*COS(2*PI*T/N));
              END;
  HANNING   : BEGIN
              WRITELN ('WAIT !  WINDOWING WITH HANNING');
              FOR T:=0 TO N-1 DO
                  X^[T].RE := X^[T].RE * (0.5-0.5*COS(2*PI*T/N));
              END;
  SQHANNING : BEGIN
              WRITELN ('WAIT !  WINDOWING WITH HANNING SQUARED');
              FOR T:=0 TO N-1 DO
```

```
                  BEGIN
                  NUM := 0.5-0.5*COS(2*PI*T/N);
                  X^[T].RE := X^[T].RE * NUM * NUM;
                  END;
              END;
  RECTANGLE : BEGIN     {no change}
              END;
  END; {CASE}
CLEARSCREEN;
END;
{***************************************************************************}
{***************************************************************************}
PROCEDURE FFT_RADIX_2 (VAR X : POINTER; GAMMA, N : INTEGER);
{performs radix 2 FFT on data pointed to by X}
CONST
    PI = 3.141592654;
VAR
    ARRAY_NUM : INTEGER;         {index of array}
    K : INTEGER;                 {index of data node in array}
    K_DUAL : INTEGER;            {index of the dual node of node K}
    DNODE_SPACING : INTEGER;     {index spacing between dual nodes in array}
    FIRST_DNODE : INTEGER;       {beginning index of next set of dual nodes}
    P : INTEGER;                 {integer power of complex exponential}
    ANGLE : REAL;                {angle argument in complex exponetial}
    ANGLE_INC : REAL;            {angle increment}
    COSINE : REAL;               {value of the cosine of the angle}
    SINE : REAL;                 {value of the sine of the angle}
    WX_REAL : REAL;              {real part product of node and complex exp.}
    WX_IMAG : REAL;              {imaginary part product of node and exp.}
BEGIN
ANGLE_INC := -2*PI/N;
DNODE_SPACING := N;
FOR ARRAY_NUM:=1 TO GAMMA DO
    BEGIN
    DNODE_SPACING := DNODE_SPACING DIV 2;   {reduced by two for each array}
    K := 0;                                 {start at top of array}
    WHILE K < N DO
        BEGIN
        FIRST_DNODE := K + DNODE_SPACING;
                       {determine power to W}
        P := DIGIT_REVERSE( K DIV DNODE_SPACING, GAMMA );
        ANGLE := P * ANGLE_INC;
        COSINE := COS( ANGLE );
        SINE := SIN( ANGLE );
        WHILE K < FIRST_DNODE DO
            BEGIN
            K_DUAL := K + DNODE_SPACING;
            WX_REAL := X^[K_DUAL].RE * COSINE - X^[K_DUAL].IM * SINE;
            WX_IMAG := X^[K_DUAL].IM * COSINE + X^[K_DUAL].RE * SINE;
            X^[K_DUAL].RE := X^[K].RE - WX_REAL;  {calculate dual nodes}
            X^[K_DUAL].IM := X^[K].IM - WX_IMAG;
            X^[K].RE := X^[K].RE + WX_REAL;    {calculate K nodes}
            X^[K].IM := X^[K].IM + WX_IMAG;
            K := K + 1;
            END;
        K := K + DNODE_SPACING;  {skip dual nodes, they're done already}
        END;
    END;
END;
{****************************************************************************}
{****************************************************************************}
PROCEDURE Derivative (VAR X : POINTER; TIMEINC : REAL; N : INTEGER);
{multiplies FFT by imaginary radian frequency.  Done when transforming
derivative of a function and not taking derivative until after transform}
VAR
    j : INTEGER;
    cnumber : COMPLEX_NUMBERS;
    freq : REAL;
BEGIN
FOR j := 0 TO N DIV 2 DO
    BEGIN
    freq := j * 2 * PI / TIMEINC / N;
    cnumber := X^[j];
    X^[j].RE := freq * cnumber.IM;
    X^[j].IM := - freq * cnumber.RE;
    END;
FOR j := (N DIV 2) + 1 TO N-1 DO
    BEGIN
    freq := (j - N) * 2 * PI / TIMEINC / N;
    cnumber := X^[j];
    X^[j].RE := freq * cnumber.IM;
    X^[j].IM := - freq * cnumber.RE;
    END;
END;
{****************************************************************************}
{****************************************************************************}
PROCEDURE PowerSpectralDensity (VAR X : POINTER;
                                    TIMEINC : REAL;
                                    N : INTEGER;
                                    VARIANCE : REAL);
{places magnitude of FFT squared in real part of complex array}
VAR
    i : INTEGER;
    energy : REAL;
    norm : REAL;
BEGIN
energy := 0;
FOR i := 0 TO N-1 DO
    BEGIN
    X^[i].RE := X^[i].RE * X^[i].RE + X^[i].IM * X^[i].IM;
    X^[i].IM := 0;
    energy := energy + X^[i].RE;
    END;
norm := VARIANCE * N * TIMEINC / energy;
FOR i := 0 TO N-1 DO
    X^[i].RE := norm * X^[i].RE;
END;
{****************************************************************************}
{****************************************************************************}
PROCEDURE AverageFFTs (VAR X : POINTER; N, iteration, numfiles : INTEGER);
{iteration is one if first FFT and indicates to store values on RAM file
without adding}
```

```
VAR
    FT : FILE OF COMPLEX_NUMBERS;
    j : INTEGER;
    cnumber : COMPLEX_NUMBERS;
BEGIN
IF iteration = 1 THEN
    BEGIN
    REWRITE (FT, 'RAM:AVETRANS');
    FOR j := Ø to N-1 DO
        WRITE (FT, X^[j]);
    END
ELSE
    BEGIN
    RESET (FT, 'RAM:AVETRANS');
    FOR j := Ø TO N-1 DO
        BEGIN
        READ (FT, cnumber);
        X^[j].RE := X^[j].RE + cnumber.RE;
        X^[j].IM := X^[j].IM + cnumber.IM;
        END;
    CLOSE (FT);
    REWRITE (FT, 'RAM:AVETRANS');
    FOR j := Ø TO N-1 DO
        WRITE (FT, X^[j]);
    END;

CLOSE (FT, 'SAVE');

IF iteration = numfiles THEN
    BEGIN
    FOR j := Ø TO N-1 DO
        BEGIN
        X^[j].RE := X^[j].RE / numfiles;
        X^[j].IM := X^[j].IM / numfiles;
        END;
    END;
END;
{****************************************************************************}
{****************************************************************************}
BEGIN  {FFT}
REWRITE(PTR,'PRINTER:');
MARK(HEAP);    {mark top of heap to later restore it to mem.}
NEW(X);
WRITE ('number of data files to transform and average?  ');
READLN (numfiles);
WRITE ('number of data points to transform?  ');
READLN (N);
GAMMA := ROUND( LN(N) / LN(RADIX) );
N := ROUND( EXP (GAMMA*LN(RADIX)) );
IF N > NMAX THEN
    N := NMAX;

FOR i := 1 TO numfiles DO
    BEGIN
    CLEARSCREEN;
    WRITE('DATA FILE NAME IS:  ');
    READLN(FILENAME);
    WRITELN (PTR, 'FOURIER TRANSFORM OF DATA FILE: ',FILENAME);

    REPEAT
        LOAD_DATA_ARRAY(X, N, TIMEINC, VARIANCE, FILENAME);
        PLOT_DATA (X, N-1, TIMEINC, N, TIMESERIES);

        IF AnswerYes ('Convolve data?') THEN
            BEGIN
            CONVOLVE (X, N);
            PLOT_DATA (X, N-1, TIMEINC, N, TIMESERIES);
            END;
        IF AnswerYes ('Window data?') THEN
            BEGIN
            WINDOW_DATA (X, N);
            PLOT_DATA (X, N-1, TIMEINC, N, TIMESERIES);
            END;
    UNTIL AnswerYes ('Finished with time domain?');

    WriteMessage ('WAIT ! CALCULATING THE FFT');
    FFT_RADIX_2 (X, GAMMA, N);
    ORDER_DATA_ARRAY (X, GAMMA, N);
    CLEARSCREEN;

    {if determining transfer function from step response data}
    IF AnswerYes ('transform step response to impulse response?') THEN
        Derivative (X, TIMEINC, N);

    {if plotting power spectral density, square FFT}
    IF AnswerYes ('plot power spectral density') THEN
        PowerSpectralDensity (X, TIMEINC, N, VARIANCE);

    IF numfiles > 1 THEN           {add new to old FFT}
        AverageFFTs (X, N, i, numfiles);

    REPEAT
        WRITE ('max frequency to plot?  ');
        READLN (MAXFREQ);
        CLEARSCREEN;
        IMAX := ROUND (MAXFREQ * TIMEINC * N);
        IF IMAX > N DIV 2 THEN
            IMAX := N DIV 2;
        PLOT_DATA (X, IMAX, TIMEINC, N, MAGNITUDE);
    UNTIL AnswerYes ('Is plot OK?');
    END;
DISPOSE (X);
RELEASE(HEAP);  {restores heap to original state to use again}
END.
```

END
DATED
FILM
8-88
DTIC